#4



PATENTS OFFICE
KILKENNY

Patents Office
Government Buildings
Hebron Road
Kilkenny

I HEREBY CERTIFY that annexed hereto is a true copy of documents filed in connection with the following patent application:

| | |
|---|---|
| Application No. | PCT/IE 01/00002 |
| Date of Filing | 8 January 2001 |
| Applicant | DELVALLEY LIMITED, an Irish company of 56 Fontenoy Street, Phibsboro, Dublin 7, Ireland. |

Dated this 24 day of July 2001.

pp An officer authorised by the
Controller of Patents, Designs and Trademarks.

HOME COPY

# PCT

## REQUEST

The undersigned requests that the present international application be processed according to the Patent Cooperation Treaty.

For receiving Office use only

International Application No. PCT/IE 01/00002

International Filing Date 08 JAN 2001

IRISH PATENTS OFFICE

Name of receiving Office and "PCT International Application"

Applicant's or agent's file reference *(if desired) (12 characters maximum)* 30907WO

**Box No. I TITLE OF INVENTION**

A data processor

**Box No. II APPLICANT**

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

DELVALLEY LIMITED
56 Fontenoy Street
Phibsboro
Dublin 7
Ireland

☐ This person is also inventor.

Telephone No.

Facsimile No.

Teleprinter No.

State *(that is, country)* of nationality: IE

State *(that is, country)* of residence: IE

This person is applicant for the purposes of: ☐ all designated States ☒ all designated States except the United States of America ☐ the United States of America only ☐ the States indicated in the Supplemental Box

**Box No. III FURTHER APPLICANT(S) AND/OR (FURTHER) INVENTOR(S)**

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

BYRNE, Michael
214 Athlumney Castle
Athlumney
Navan
County Meath
Ireland

This person is:

☐ applicant only

☒ applicant and inventor

☐ inventor only *(If this check-box is marked, do not fill in below.)*

State *(that is, country)* of nationality: IE

State *(that is, country)* of residence: IE

This person is applicant for the purposes of: ☐ all designated States ☐ all designated States except the United States of America ☒ the United States of America only ☐ the States indicated in the Supplemental Box

☒ Further applicants and/or (further) inventors are indicated on a continuation sheet.

**Box No. IV AGENT OR COMMON REPRESENTATIVE; OR ADDRESS FOR CORRESPONDENCE**

The person identified below is hereby/has been appointed to act on behalf of the applicant(s) before the competent International Authorities as: ☒ agent ☐ common representative

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country.)*

O'CONNOR Donal H
SCHUTTE Gearoid

of

CRUICKSHANK & CO., 1 Holles Street, Dublin 2, Ireland

Telephone No.
(+353) 1 6612533

Facsimile No.
(+353) 1 6612480

Teleprinter No.

☐ **Address for correspondence:** Mark this check-box where no agent or common representative is/has been appointed and the space above is used instead to indicate a special address to which correspondence should be sent.

**Continuation of Box No. III** **FURTHER APPLICANT(S) AND/OR (FURTHER) INVENTOR(S)**

***If none of the following sub-boxes is used, this sheet should not be included in the request.***

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

GOMEZ, Maribel
Becherstr. 44
D-40476 Dusseldorf
Germany

This person is:
- ☐ applicant only
- ☒ applicant and inventor
- ☐ inventor only *(If this check-box is marked, do not fill in below.)*

State *(that is, country)* of nationality: ES

State *(that is, country)* of residence: DE

This person is applicant for the purposes of:
- ☐ all designated States
- ☐ all designated States except the United States of America
- ☒ the United States of America only
- ☐ the States indicated in the Supplemental Box

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

MOORE, Thomas
6 Ardee Grove
Ardee Road
Rathmines
Dublin 6
Ireland

This person is:
- ☐ applicant only
- ☒ applicant and inventor
- ☐ inventor only *(If this check-box is marked, do not fill in below.)*

State *(that is, country)* of nationality: IE

State *(that is, country)* of residence: IE

This person is applicant for the purposes of:
- ☐ all designated States
- ☐ all designated States except the United States of America
- ☒ the United States of America only
- ☐ the States indicated in the Supplemental Box

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

O'RIORDAN, Martin
Bryanstown Avenue
Ladychapel
Maynooth
County Kildare
Ireland

This person is:
- ☐ applicant only
- ☒ applicant and inventor
- ☐ inventor only *(If this check-box is marked, do not fill in below.)*

State *(that is, country)* of nationality: IE

State *(that is, country)* of residence: IE

This person is applicant for the purposes of:
- ☐ all designated States
- ☐ all designated States except the United States of America
- ☒ the United States of America only
- ☐ the States indicated in the Supplemental Box

Name and address: *(Family name followed by given name; for a legal entity, full official designation. The address must include postal code and name of country. The country of the address indicated in this Box is the applicant's State (that is, country) of residence if no State of residence is indicated below.)*

This person is:
- ☐ applicant only
- ☐ applicant and inventor
- ☐ inventor only *(If this check-box is marked, do not fill in below.)*

State *(that is, country)* of nationality:

State *(that is, country)* of residence:

This person is applicant for the purposes of:
- ☐ all designated States
- ☐ all designated States except the United States of America
- ☐ the United States of America only
- ☐ the States indicated in the Supplemental Box

☐ Further applicants and/or (further) inventors are indicated on another continuation sheet.



*See Notes to the request form*

**Box No.V DESIGNATION OF STATES**

The following designations are hereby made under Rule 4.9(a) *(mark the applicable check-boxes; at least one must be marked)*:

**Regional Patent**

- [X] **AP** **ARIPO Patent:** GH Ghana, GM Gambia, KE Kenya, LS Lesotho, MW Malawi, SD Sudan, SL Sierra Leone, SZ Swaziland, TZ United Republic of Tanzania, UG Uganda, ZW Zimbabwe, and any other State which is a Contracting State of the Harare Protocol and of the PCT
- [X] **EA** **Eurasian Patent:** AM Armenia, AZ Azerbaijan, BY Belarus, KG Kyrgyzstan, KZ Kazakhstan, MD Republic of Moldova, RU Russian Federation, TJ Tajikistan, TM Turkmenistan, and any other State which is a Contracting State of the Eurasian Patent Convention and of the PCT
- [X] **EP** **European Patent:** AT Austria, BE Belgium, CH and LI Switzerland and Liechtenstein, CY Cyprus, DE Germany, DK Denmark, ES Spain, FI Finland, FR France, GB United Kingdom, GR Greece, IE Ireland, IT Italy, LU Luxembourg, MC Monaco, NL Netherlands, PT Portugal, SE Sweden, and any other State which is a Contracting State of the European Patent Convention and of the PCT
- [X] **OA** **OAPI Patent:** BF Burkina Faso, BJ Benin, CF Central African Republic, CG Congo, CI Côte d'Ivoire, CM Cameroon, GA Gabon, GN Guinea, GW Guinea-Bissau, ML Mali, MR Mauritania, NE Niger, SN Senegal, TD Chad, TG Togo, and any other State which is a member State of OAPI and a Contracting State of the PCT *(if other kind of protection or treatment desired, specify on dotted line)* .....

**National Patent** *(if other kind of protection or treatment desired, specify on dotted line)*:

- [X] AE United Arab Emirates
- [X] AL Albania
- [X] AM Armenia
- [X] AT Austria
- [X] AU Australia
- [X] AZ Azerbaijan
- [X] BA Bosnia and Herzegovina
- [X] BB Barbados
- [X] BG Bulgaria
- [X] BR Brazil
- [X] BY Belarus
- [X] CA Canada
- [X] CH and LI Switzerland and Liechtenstein
- [X] CN China
- [X] CR Costa Rica
- [X] CU Cuba
- [X] CZ Czech Republic
- [X] DE Germany . and . Utility . Model
- [X] DK Denmark . and . Utility . Model
- [X] DM Dominica
- [X] EE Estonia
- [X] ES Spain
- [X] FI Finland
- [X] GB United Kingdom
- [X] GD Grenada
- [X] GE Georgia
- [X] GH Ghana
- [X] GM Gambia
- [X] HR Croatia
- [X] HU Hungary
- [X] ID Indonesia
- [X] IL Israel
- [X] IN India
- [X] IS Iceland
- [X] JP Japan
- [X] KE Kenya
- [X] KG Kyrgyzstan
- [X] KP Democratic People's Republic of Korea
- [X] KR Republic of Korea
- [X] KZ Kazakhstan
- [X] LC Saint Lucia
- [X] LK Sri Lanka
- [X] LR Liberia
- [X] LS Lesotho
- [X] LT Lithuania
- [X] LU Luxembourg
- [X] LV Latvia
- [X] MA Morocco
- [X] MD Republic of Moldova
- [X] MG Madagascar
- [X] MK The former Yugoslav Republic of Macedonia
- [X] MN Mongolia
- [X] MW Malawi
- [X] MX Mexico
- [X] NO Norway
- [X] NZ New Zealand
- [X] PL Poland
- [X] PT Portugal
- [X] RO Romania
- [X] RU Russian Federation
- [X] SD Sudan
- [X] SE Sweden
- [X] SG Singapore
- [X] SI Slovenia
- [X] SK Slovakia
- [X] SL Sierra Leone
- [X] TJ Tajikistan
- [X] TM Turkmenistan
- [X] TR Turkey
- [X] TT Trinidad and Tobago
- [X] TZ United Republic of Tanzania
- [X] UA Ukraine
- [X] UG Uganda
- [X] US United States of America
- [X] UZ Uzbekistan
- [X] VN Viet Nam
- [X] YU Yugoslavia
- [X] ZA South Africa
- [X] ZW Zimbabwe

Check-boxes reserved for designating States which have become party to the PCT after issuance of this sheet:

- [ ] .....
- [ ] .....

**Precautionary Designation Statement:** In addition to the designations made above, the applicant also makes under Rule 4.9(b) all other designations which would be permitted under the PCT except any designation(s) indicated in the Supplemental Box as being excluded from the scope of this statement. The applicant declares that those additional designations are subject to confirmation and that any designation which is not confirmed before the expiration of 15 months from the priority date is to be regarded as withdrawn by the applicant at the expiration of that time limit. *(Confirmation (including fees) must reach the receiving Office within the 15-month time limit.)*

PCT/IE 01/00002

**Box No. VI PRIORITY CLAIM** ☐ Further priority claims are indicated in the Supplemental Box.

| Filing date of earlier application (day/month/year) | Number of earlier application | Where earlier application is: national application: country | regional application:* regional Office | international application: receiving Office |
|---|---|---|---|---|
| item (1) 28/07/2000 | S2000/0603 | IE | | |
| item (2) | | | | |
| item (3) | | | | |

☒ The receiving Office is requested to prepare and transmit to the International Bureau a certified copy of the earlier application(s) *(only if the earlier application was filed with the Office which for the purposes of the present international application is the receiving Office)* identified above as item(s): Item (1)

* *Where the earlier application is an ARIPO application, it is mandatory to indicate in the Supplemental Box at least one country party to the Paris Convention for the Protection of Industrial Property for which that earlier application was filed (Rule 4.10(b)(ii)). See Supplemental Box.*

**Box No. VII INTERNATIONAL SEARCHING AUTHORITY**

**Choice of International Searching Authority (ISA)** *(if two or more International Searching Authorities are competent to carry out the international search, indicate the Authority chosen; the two-letter code may be used):* ISA / EP

**Request to use results of earlier search; reference to that search** *(if an earlier search has been carried out by or requested from the International Searching Authority):*

Date *(day/month/year)* | Number | Country *(or regional Office)*

**Box No. VIII CHECK LIST; LANGUAGE OF FILING**

This international application contains the following **number of sheets:**

| | |
|---|---|
| request | 4 |
| description (excluding sequence listing part) | 34 |
| claims | 7 |
| abstract | 1 |
| drawings | 9 |
| sequence listing part of description | |
| **Total number of sheets** | 55 |

This international application is **accompanied by** the item(s) marked below:

1. ☒ fee calculation sheet
2. ☐ separate signed power of attorney
3. ☐ copy of general power of attorney; reference number, if any:
4. ☐ statement explaining lack of signature
5. ☐ priority document(s) identified in Box No. VI as item(s):
6. ☐ translation of international application into (language):
7. ☐ separate indications concerning deposited microorganism or other biological material
8. ☐ nucleotide and/or amino acid sequence listing in computer readable form
9. ☐ other *(specify)*:

**Figure of the drawings which should accompany the abstract:** 1

**Language of filing of the international application:** English

**Box No. IX SIGNATURE OF APPLICANT OR AGENT**

*Next to each signature, indicate the name of the person signing and the capacity in which the person signs (if such capacity is not obvious from reading the request).*

C. Schutte

SCHUTTE Gearoid

**For receiving Office use only**

1. Date of actual receipt of the purported international application: 08 JAN 2001
2. Drawings: ☒ received: ☐ not received:
3. Corrected date of actual receipt due to later but timely received papers or drawings completing the purported international application:
4. Date of timely receipt of the required corrections under PCT Article 11(2):
5. International Searching Authority (if two or more are competent): ISA / EP
6. ☐ Transmittal of search copy delayed until search fee is paid.

**For International Bureau use only**

Date of receipt of the record copy by the International Bureau:



*See Notes to the request form*

# "A Data Pr c ssor"

## Introduction

The present invention relates to a data processor and in particular to a data processor of the Reduced Instruction Set Computer (RISC) type data processor.

As the computational requirements of data processing increase the datapath widths of the processors have correspondingly tended to increase. Typically, currently used data processors are 16-bit, 32-bit and 64-bit processors i.e. having datapath widths of 16-bit 32-bit and 64-bit datapaths. Further the number of registers within the data processors have increased not alone in size because of the datapath width, but also in number because of the complexity and of the computations.

Essentially when a data processor is being designed, the first thing that happens is that the various computational and other requirements of the processor are specified in a program. Then the designer, or programmer specifies the requirements of the data processor to tackle this task, specifying the number of bits of datapath width required, the number of registers, memory and other computational requirements. Such a processor, which will contain at least a configurable logic unit, a plurality of registers and accessible memory and the various datapaths between the components. Having done this the programmer will then choose some processor and will then specify that processor which will then be embodied in silicon. The first problem that arises for the designer is that very often he or she has to make a choice between a 32-bit, 64-bit or other standard size processor. Suppose, for example, the requirement is actually for something with a 37-bit datapath width, 10 registers and a certain memory and logical unit capacity. The designer has a first choice as to whether he or she will choose a 32-bit dataprocessor and use it, or a 64-bit data processor. If a 32-bit data processor is used, then it may be slower than a 64-bit data processor, but almost certainly the latter will cost substantially more and the chip embodying the processor will also be substantially larger in size, probably of the order of 100%. If then the only processor that the designer can get is one with an excess capacity of registers, then the chip being manufactured will also have a considerable amount of redundant space.

Further problems arise with the increase in datapath width in that the registers within the processors have also increased to have matching widths and many data being processed will be smaller than the datapath widths and thus large registers are used to store words in a wasteful manner. At the same time it is appreciated that the greater the number of dataprocessing registers available, then the more data can be stored in registers with fewer reads from or writes to cache or main memory. The disadvantage of providing a larger number of registers is the complexity and costs increase and as mentioned already increasing the size of the registers enabling them to store or manipulate a larger amount of data has the resultant disadvantages of cost, increased complexity and physical size.

RISC pipelining architecture has in general produced an increase in speed of processing to one command per processor system clock cycle. One particular model, known as the Harvard model, is used in such processors and has in many instances replaced the previously used von Neumann model. In the Harvard model the storage areas are separated and accessed by using different access routes. In both of these cases processing and result sequencing of the command flow is carried out.

Generally it has been realised that what is required is a processor that could be effectively infinitely configurable. What is needed is a generic processor. It will be appreciated that practically not every component of the processor needs to be infinitely variable. Various attempts have been made to do this, but heretofore have been relatively unsuccessful. Essentially what is required is a processor that can be specified exactly down to all the various components, whether they be the datapath width, memory, number of registers, size of registers and so on so that a designer can specify exactly the size and configuration of chip required to carry out the particular processing tasks. Thus, what is required is a processor with no redundant components either in number or size.

For example, U.S. Patent Specification No. 6,061,367 (Siemens) discloses a processor having a pipeline architecture and a configurable logic unit. This processor includes as well as the configurable logic unit, an instruction memory, a decoder unit, an interface device, a programmable structure buffer, an integer/address instruction buffer and a multiplex-controlled s-paradigm unit linking contents of an integer register file to a functional unit with programmable structures and having a large number of data links connected by multiplexers. The s-paradigm unit has a programmable hardware structure for dynamic

reconfiguration/programming while the program is running. The functional unit has a plurality of arithmetic units for arithmetic and/or logic linking of two operands on two input buses to produce a result on an output bus, a plurality of compare units having two input buses and one output bit, a plurality of multiplexers having a plurality of input buses and one or two output buses and being provided between the arithmetic units, the compare units and the register file, and a plurality of demultiplexers having one input bit and a plurality of output bits. A method is also provided for high-speed calculation with pipelining.

Various other attempts have been made to provide improved constructions of processors, such as, for example, those produced by the company Arm Limited. Typical examples of their processors are described in various U.S. Patent Specifications. For example, U.S. Patent Specification No. 5,969,975 (Arm) attempts to overcome the disadvantages of the complexity and increase in number of registers by providing an arithmetic logic unit to receive input operands from M X-bit registers to produce output datawords stored within N Y-bit registers, where M/N = 3, $8 \leq Y\text{-}X \leq 16$ and $3X=2Y$. It is suggested that this arrangement is particularly suitable for digital signal processing and in situations where each input operand is used a plurality of times before a new input operand is loaded in its place in a register.

U.S. Patent Specification No. 5,881,259 (Arm) is directed to accessing a memory having a plurality of memory locations for storing data values and in particular to a data processor that prevents memory access.

U.S. Patent Specification No. 6,021,476 (Arm) again is directed towards the accessing of memory in data processors.

U.S. Patent Specification No. 5,961,633 (Arm) provides a data processor in which successive data processing instructions are again executed in a pipeline architecture. This processor contains conditional control means for preventing complete execution of a current instruction if either the memory detects that a memory access initiated by a preceding instruction is invalid, or if in some way it detects that the current instructions should not be executed.

U.S. Patent Specification No. 5,132,898 (Mitsubishi Denki Kabushiki Kaisha) describes another type of processor for carrying out operations between operands having different bit lengths of data and it illustrates very clearly the problems involved in the manipulation of such data.

While various attempts have been made, as mentioned already, to provide configurable processors, a considerable amount of the activity involved has been in improving the operation of processors generally and in improving their architecture without in fact tackling the major problem which is that what the user wants is a processor directed entirely towards the task in hand i.e. to allow the programmer or designer produce a processor, which processor will have a specification ideally matched to the processing requirements. Once this has been done then a considerable amount of the problems in relation to actual processing operations, etc. become less relevant.

## Statements of Invention

According to the invention there is provided a processor having a number of components including at least a configurable arithmetic and logic unit, a plurality of registers, memory access, and datapaths between the components, characterised in that:

the datapath width is of variable bit size namely n bits;

the number of the components are selectable;

where appropriate the components are of n bit size; and

each component is configured to handle data having one of two sizes

$$\leq n \quad \text{or} \quad > n.$$

While the number of components is arbitrarily chosen, this is largely done for optimisation of the processor, but the processor essentially could have a components, where a was any number. It is often found in practice for example that producing 32 registers in the actual design of processor is an adequate number of registers for some particular uses. There is

however no reason why additional or less registers could not be produced and prepared. Many of the components will have to be of the n bit size to match the datapath width, but other of the components need not. For example, it is envisaged that the registers can be specified to any bit size, thus overcoming the problems as mentioned already in relation to register sizes. It is however important to appreciate that any input can be greater than or less than the datapath width size. Particularly this may be the case with memory where memory sizes will be larger than the datapath width. Further it may be that under normal operating conditions, the processor may be required to process data of 73 bits in length. The situation may arise where the processor is required to handle data of, for example, 150 bits in length in rare situations. In this scenario, instead of designing a processor with a datapath width of 150 bits, the designer could design an optimal processor having a datapath width of 73 bits and program the processor to be able to handle the 150-bit piece of data as that situation arises. This will help to avoid redundancy under normal operating conditions.

Ideally such a processor comprises:

means to select the number and size of each component;

means to select the datapath width;

means to configure the components for that datapath width; and

means to compare the width of a data input to the selected datapath width that has been chosen for the component.

By having such a processor, once the designer has specified the requirements, it is possible for the designer then to simply take the processor according to the present invention and input the various data. Then having inputted the various data requirements, such as, for example, in a database or other document, the processor can be used to effectively provide the processor and make it in silicon. What has been designed is a processor that will allow the developer to mould it to the need at hand.

It will be appreciated therefore to a certain extent what is being provided according to the

present invention is not so much a processor, but in fact a template to allow a processor to be produced, in the sense that there will never be produced a processor of n bits wide. What will be produced for example is a processor with a datapath width of 37 with for example 15 registers, a configurable arithmetic and logic unit containing the logic required and memory access. This will then be realised in silicon, which will also mean that it will be as quick as using a standard 64-bit processor and only marginally more bulky and costly than a 32-bit processor. If fewer registers were a requirement, it might be less costly and bulky than an off-the-shelf 32-bit processor.

In one embodiment of the invention when the immediate data of an instruction is limited in size to a preset number of bits and this number is less than n the immediate data is expanded to n bits wide. However, when the immediate data of each instruction is greater than n, then the immediate data has to be truncated.
Ideally the processor has special purpose registers and then general purpose registers. The general registers are dependent on the bit size of the data being handled and will be of size n bits, but not all of the special registers need to be of size n bits.

It is envisaged that the general registers may be mounted external of the processor and the processor according to the invention is so-configured and thus all that a designer requires is to specify those registers to be held external. Also, most of the special registers can be mounted externally.

In a particular embodiment of the invention the registers are configured to allow their content to be written to memory external of the processor. In this way in certain situations the special registers can have two functions, which further reduces the size of the processor. They will act as general registers when required and will still be able to act as special registers.

In some instances, all the general registers will indeed be n bits wide.

It is to be appreciated that data items of sizes other than n can be passed into the datapath of width n. In the processor, means are provided for extending or truncating a data item of size x so that it matches the width n of the datapath. In the case of truncation, that is where x is greater than n, the data item of size x is truncated the size n with the most significant

end being discarded. If the data item of size x is less then n, the data item needs to be extended, how this is to be extended depends on two situations. The first situation is where the sign of the data item is to be maintained, here the $(x-1)^{th}$ bit is replicated into bit x through to the $(n-1)^{th}$ bit, basically padding out the data item so that it fits the datapath width. The second situation is where no sign extension is required. In this situation, the data item of size x is padded out with zeros in the same range of bit locations as with the signed situation, until it is of width n. The only other case is where x is equal to n. Here there is no extension or truncation required so the data item of size x is passed straight into the datapath without any alterations.

Means are provided in the processor to perform logical operations on different halves of operands within the processor. Two different types of these half operand logical instructions are available. The X type operations swap the upper and lower halfwords of the first source operand and then perform the bitwise logical operation specified between this swapped operand and the second operand. The second type, S type operations, perform the bitwise logical operation specified on the two source operands, the upper and lower halfwords of the result are then swapped before it is passed on through the processor pipeline. The bitwise logical instructions that these type of operations involve are AND, NAND, OR, NOR, XOR and XNOR resulting in ANDS, NANDS, ANDX, NANDX, ORS, NORS, ORX, NORX, XORS, XNORS, XORX, XNORX and the immediate instruction equivalent versions.

The processor, according to the present invention is designed and arranged so that separate functions to perform special logic operations can be added as separate units. These units will have been developed separate from the processor. However, the processor provides a single interface structure that presents common signals to all of these separate units thus enabling any one or any number of units to be added. This single interface is fixed providing 3 outputs that contain an operation code identifier (aluOp) and two operands (aluS1 and aluS2) to perform the selected operation on. Also provided for are two inputs, one containing the result of the selected operation and the other a signal to indicate when the result is valid. Within the processor itself, as these separate functions are added, so the ability of the processor to determine that these functions are to be used is developed, thus, when the processor is instructed to perform these separate operations, it will do so. These separate units can execute in one or more clock cycles (multicycle

operation) and are integrated into the control logic of the microprocessor to the extend that stall control and data forwarding is performed identically to the way it is performed for the built in units.

The processor according to the present invention can be so-arranged that both sets of registers can be shared between various processors. Thus, for example, in certain situations when more than one processor would be required in a particular application in the sense that while the designer or programmer might require two or more processors, that in the specifying of those processors it would be possible to use the same registers for both processors.

It is envisaged that the processor according to the present invention will be embodied in a computer disk or the like storage medium and can be simply downloaded by an operator, the various parameters inputted, the processor configured and then downloaded for subsequent manufacture in silicon or the like material.

Further the invention provides a method of designing a processor comprising the steps of:

preparing an outline processor in general architecture having a series of components described by blocks or the like interconnected by various datapaths having at least a configurable arithmetic and logic unit, a plurality of registers, memory access, and such other units and components as are required for a processor of the type being designed and then defining the datapath width of variable bit size namely n bits;

choosing an arbitrary number of components greater than that which would ever be required such as, for example, 64 registers; or alternatively

choosing components where a is any number that could be chosen; defining the components size as n bit size; and

programming each component to handle data having one of two sizes, namely $\leq n$ or $> n$.

In this way a general processor is designed and then subsequently when it is required to produce a processor from this general design the number of components, the datapath width size and so on are chosen and they are entered into a database, which database will allow a particular design of processor to be produced.

**Detailed Description of the Invention**

The invention will be more clearly understood from the following description of an embodiment thereof given by way of example only with reference to the accompanying drawings, in which:

Fig. 1 is a block diagram of a processor according to the invention and the external interfacing;

Fig. 2 illustrates the basic processor pipeline;

Fig. 3 illustrates the basic processor pipeline with control signals;

Fig. 4 is a block diagram illustrating a bitwise logic X instruction;

Fig. 5 is a block diagram illustrating a bitwise logic S instruction;

Fig. 6 illustrates the processor pipeline in more detail;

Fig. 7 is a block diagram of the processor information;

Fig. 8 is a flow diagram illustrating the data memory sign extend unit;

Fig. 9 is an overall block diagram of the register unit;

Fig. 10 is a block diagram of the general purpose registers; and

Fig. 11 is a block diagram of the register multiplexers (muxes).

Referring now to Fig. 1 there is illustrated in block diagrammatic form an outline of the processor according to the invention and the external interfacing to it. All of the external interfacing has various signals to and from the processor. The processor is identified by the reference numeral 1 and the principal components illustrated are instruction decoding 2 which in turn feed an arithmetic logic unit (ALU) 4 through datapaths 5 of n bits wide. Further datapaths 5 are also illustrated as is a data memory control 6 fed from the arithmetic logic unit 4. The data memory control 6 also feeds the general purpose and special registers which together with the instruction decoding 2 also feed the arithmetic logic unit 4 through a mux 7. Signal descriptions for Fig. 1 are listed below and are elaborated on somewhat later.

***sysClk***

This is the system wide clock provided to the processor. All pipelining and registering within the processor is done on this clock.

***sysReset***

This is the reset signal provided by the system. It is active high .

***imAddr[m-1:0]***

This is the instruction memory address bus. It can be synchronous or asynchronous. It is in byte address sizes but all values that appear on it are word addresses. On a reset this bus goes to zero. M is the configured program memory address width.

***imData[p-1:0]***

This is the data from the instruction memory i.e. it is the instruction addressed by the instruction memory address bus. P is the configured size of the instruction data.

***imRdy***

This signal indicates when valid data is available from the instruction memory. If the instruction memory takes more than a clock cycle to produce valid data from when it is addressed, this signal must be pulled low until valid data is available.

***dmAddr[q-1:0]***

During accesses to data memory, the address of the data location appears on this bus. It is a registered output. The addresses that appear on this bus are byte addresses.

***dmDataIn[n-1:0]***

If a Load from memory instruction occurs, the data from the data memory location, addressed by *dmAddr[q-1:0]*, is passed to the processor on this bus.

***dmDataOut[n-1:0]***

If a Store to memory instruction occurs, the data to be written to the data memory location, addressed by *dmAddr[q-1:0]*, appears on this bus.

***dmCS***

When this signal is HIGH it indicates that an access to memory is occurring

***dmRW***

This signal indicates to the memory whether a load or store is happening. If it is HIGH this indicates a store to memory and if it is LOW a load from memory is happening.

***dmSiz[1:0]***

This output signal is used by the processor to indicate to the data memory when word, halfword or byte transfers are required. **b00** indicates that the transfer is a byte, **b01** indicates that the transfer is a halfword and **b10** indicates that the transfer is a word. These values are valid for both loads and stores.

***dmRdy***

This input signal indicates when valid data is available from the data memory. If the data memory takes more than a clock cycle to produce valid data from when it is addressed, this signal must be pulled low until valid data is available.

***extInt***

This input signal is the request from an external device to interrupt the processor. It must be held high for at least 1 *sysClk* clock cycle.

***extIntAck***

When the processor receives the external interrupt and starts to service the interrupt, this signal is set high for 1 *sysClk* clock cycle to acknowledge the interrupting source that it has received the interrupt.

As explained the architecture of the processor is based around the Harvard architecture model. This model includes the non-sharing of instruction and data memory space which lends itself to a very low cycle per instruction count as there is no contention for memory. Potentially if there is zero wait memory, such as asynchronous SRAM, the processor will not have to stall and wait for any memory access to be completed. Essentially the processor according to the present invention is shown in five stages. This is illustrated in Fig. 2

The pipelining technique allows the overlapped execution of multiple instructions. The pipeline in the present processor is divided into five stages. All of the stages use the same clock cycle so an instruction is completed every clock cycle and the duration of an instruction is five clock cycles. It will be appreciated therefore that this is a particularly suitable form of processor as the through-put is increased by a factor of five, under ideal conditions. It is important to appreciate that all the stages are active on every clock cycle.

Referring now to Fig. 2 the elements of each stage of the pipeline is described in somewhat more detail with the memory connected thereto. The processor is again indicated by the reference numeral 1 and the stages are divided into five stages, namely a Fetch stage 10, a Decode stage 20 , an Execution stage 30, a Load and Storage stage 40 and a Write Back stage 50. Because the stages are identified by different reference numerals, the components previously identified by a reference numeral now may have a different reference numeral attached thereto. The Fetch stage 10 implements the loading of the next instruction to be executed. A program counter (PC) keeps track of the instruction number to be executed. The Fetch stage 10 includes an instruction memory 11 and address buses connected to this instruction memory 11 and a multiplexer (mux) 12 to select the next PC. The memory is addressed by the actual value of PC and the content of that position is registered and sent to the decode stage 20. The multiplexer 12 selecting

the next PC is dependent on the instruction being decoded at the same clock cycle in the decode stage. It determines whether to choose from the PC +4 or the target address for branch or jump instructions. It is passed out as the instruction memory address and the data returned. In the Decode stage 20 after the instruction has been passed from memory the Decode stage 20 decodes the instruction to determine the operation to be performed in operands that are selected by the instruction. These operands are from registered address by the instruction, or a value provided by the instruction. This is where the whole control of the whole pipeline occurs. It takes the instruction from the Fetch stage 10 and decodes it in order to set the signals which will control its execution. Part of the information present in the instruction being decoded are the addresses of the registers involved in some operations. There is thus provided a decoder 21, general purpose and special purpose registers 22, a sign extend unit 23 and a multiplexer 24 for selecting the next PC. Part of the information present in the instruction which is being decoded in the decoder 21 are the addresses of the registers 22, thus they address the source operands of the general purpose and special registers 22 and their contents are registered to become the inputs for the Execution stage 30. In the case of an immediate operation, the 16-bit immediate value, which is the usual value coming from the instruction is either sign extended or padded with zeroes in the sign extender 23. This sign extend unit 23 is a dedicated sign extend unit that will be described in some more detail below. Generally speaking the instruction data will be in 32-bits with 16-bit immediate value. The processor can be configured for higher inputs, but they are not generally required and thus in the description of the processor there is this limitation.

When decoding a branch instruction or a jump, the value of next PC is appropriately changed. Decisions of whether to change or not to change the value of the PC and the calculation of the target address are done in the Decode stage 20 by means of control logic and an additional adder. In the case of TRAP, RET or RFE instructions the PC is also changed from the normal flow to a predetermined value.

The Execution stage 30 is where the actual implementation of the operation decoded in the Decode stage 20 is performed. This is where an ALU unit 31 is illustrated which ALU unit 31 is in fact the ALU 4 already identified in Fig. 1. In the Execution stage 30 the ALU operation indicated in the instructions and registers is performed and delivered to the next stage of the pipeline. It calculates the address for the data memory access in

the Load/Store stage 40 which will be performed in the next clock cycle in the Load/Store stage 40. The source operand could be either a register or an immediate. Thus, there is a multiplexer 32 provided to decided between them.

The next stage is the Load/Store stage that also could be called the memory stage 40. The data memory address 41 and data buses, as well as the corresponding control signals in turn has a further multiplexer 42 to all the either the memory data or ALU result to be registered in what is effectively the last stage, which is the Write Back stage 50. It passes data to be written to the general purpose registers or special purpose registers 22 and the control signal to do it.

The above is a brief outline of the architecture. It does not describe it in great detail and indeed most of the architecture can be said to be essentially conventional.

However, the processor according to the present invention is extensively configurable and parameterizable. The datapath width has been set at n bits and the number of registers and the size of instruction and data memories accessible are configurable.

The data length of the datapath elements and almost all the registers of the processor can be configured to any width from 1 bit to n bits, namely a word length of n bits for the processor. The processor according to the invention also uses data of two other sizes, namely halfwords and bytes. Halfwords are half the width of the word length, needless to say if the word length is an odd length, namely n is not an even number, the half word is modulus of half the width of the word length. Sometimes in the following discussion the term byte, which is 8-bits, is used, but will be understood by those skilled in the art.

According to the invention the width and amount of registers within the processor may be configured. Again this is described in more detail. For ease of design and use, it is normal to pick a maximum number of registers according to the invention, such as, for example, 64 registers and to design the processor for 64 registers. Thus, generally speaking the registers, except for some are of n bits wide and the actual number of registers is arbitrarily chosen in due course as will be explained later. The important point to appreciate is that all these registers are provided which may be configured as required.

In the particular processor according to the present invention the instruction and data memories are physically outside of the processor, however, the amount of memory accessible is defined by the processor. Both the instruction memory address, *imAddr*, and the data memory address, *dmAddr*, are generated by the processor and the width of these busses can be set to match the size of memory needed (see Fig. 1).

The data width of these memories can also be configured with the data memory width matching the width of the processor datapath width, namely n.

In relation to the instruction memory while this instruction memory width is determined by the width of the instructions, the processor according to the present invention is so-arranged that the instruction memory width can be variable. However, at the present moment because it has been found that an instruction width of 32-bits is sufficient for the present design so it is carried out with the instruction width fixed at 32-bits wide. Obviously this has the ability to be changed if the instruction memory use some form of compression or alternatively could be extended.

Various other configurations of the processor are included, for example, interrupts can be enabled or disabled, the number of interrupts required can be configured, special hardware functions can be added as special ALU operations. Further instructions which are derived from the instructions of the processor can also be added. Again, this is discussed in more detail below.

Reference has already been made to the registers and they have been described as both general purpose registers and special registers. The general purpose registers (GPR) are the set of registers that can be read or written to by all instructions that access registers. Register R0 always returns 0. The number of GPRs in the processor can be configured in this particular embodiment up to a maximum of 32 and the width of the registers match the configured datapath width n.

The special registers are a second set of registers in the processors. These registers can be read or written by instructions that perform an operation where the two source operands are register values. The first four registers are used for controlling the

processors and these four registers are a reason register, link address register, exception address register and an interrupt register. It is possible to configure up to 32 registers as with the GPRs. The reason register explains the present state of the processor 4 bits are used and generally they are as listed below.

| n | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|
| Reserved | | I | T | O | R |

where:

R – indicates that the processor has been powered up from a full hard system reset.

O – If the processor received an illegal instruction this bit will be set and the processor will start executing from the start address again. It also will have the effect of clearing the R bit so that it is indicated that the last reset was a soft reset as opposed to a hard system reset. If this bit has been set and there then is a hard reset, this bit will be cleared.

T – If a trap instruction has been encountered, the processor will execute an exception service routine and while this routine is being executed, this bit is set. On exiting the exception service routine, this bit will be cleared.

I – This has the same operation as the T bit except it is set while an exception service routine is being executed as a result of an external interrupt.

The link address register contains the value of the return address when the code being executed jumps to another instruction and intends to return back to the original section of code. An example of this is a procedure call. The width of this register has a minimum size of the instruction memory address width a if the datapath width n is less than a however, if the datapath width is greater than the instruction memory address width a, this register takes on the size of the datapath n.

The exception address register is at register address 2 in the special registers set. This register contains the value of the return address when the code being executed jumps to another instruction and intends to return back to the original section of code. The instructions that cause it are JAL and JALR. If those instructions are not present in the code, this register can be used as Special Register otherwise the return address will be overwritten.

The interrupt register is at register address 1 in the special registers set. This register is n bits wide with the bottom half of the register holding the enable bits and the top half containing the pending bits.

This register and support logic controls the interrupt handling of the processor. As this register matches the datapath width and two bits of the register are used per interrupt, then the number of allowable interrupts is n/2. When an interrupt is received the pending bit is set. Then if the enable bit is set, the processor will automatically service the interrupt.

Both the General Purpose Registers and the Special Registers can exist either inside the processor or outside it, depending on the configuration required. In this implementation, the first four Special Registers are always inside the processor, the rest of the Special Registers can be either inside or outside it. All the GPR's can be either inside or outside the processor.

In the present implementation of the processor, there are four instruction formats . As all the opcodes have not been used, many more additional instructions may be introduced.

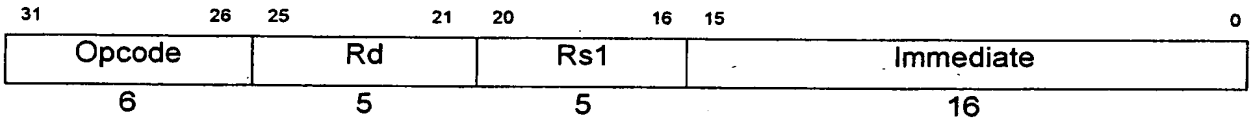
In the present processor the instructions have initially been implemented at 32-bit wide. However, this has been set as a parameter of bit wide p, which can be changed if a reduction or expansion in instruction memory width is implemented and some form of Fetch stage decompression is used to expand the instruction to its intended size.

The first format is an I-type (immediate) instructions which manipulates data provided by a 16-bit signed or unsigned immediate field in the instruction. These immediate instructions break down as follows:

- Immediate ALU operations where the immediate is used as an operand for the ALU and the result is written back to a register.

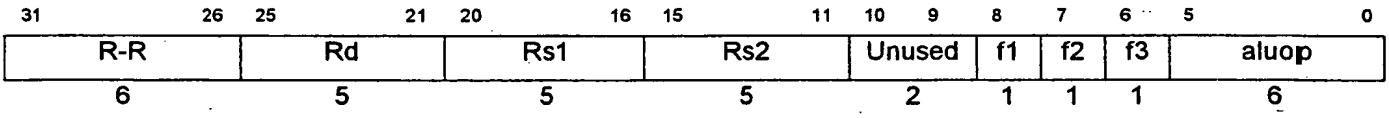
- Conditional branch instructions where the immediate is added as an *offset* to the Program Counter to transfer control of the processor to a different point in the source code.

- Load from Memory and Store to Memory instructions use the immediate data as the offset to a register value to generate the memory address to be accessed.




The immediate data of an instruction or the data from memory may be in either signed or unsigned binary format. If the data is in signed format, then it is imperative that the sign be maintained if the data should go through any expansion. The processor handles this by firstly determining whether or not a piece of data is in signed or unsigned format. If the data is in unsigned format, then the processor will populate the vacant bits of the datapath with zeroes. If the data should happen to be in signed format, then the vacant bit positions of the datapath up to the $(n-1)^{th}$ bit are populated with the MSB of the data. Generally speaking, these will be populated with ones should the data be negative signed binary, and zeroes should the data be positive signed binary. Sign expansion will be discussed in more detail below.

The second format of instruction is the R-type (register to register) instructions which perform pure ALU type operations on two operands provided by two source registers specified in the instruction. The result is always destined for a register. The operation to be performed is specified by the aluop field of the instruction. Access to the Special Register set from source code can only happen through R-type instructions except for Special Register 1 (Link Address Register). Special Registers are identified by 3 1-bit flags in the instruction and are shown and explained below.



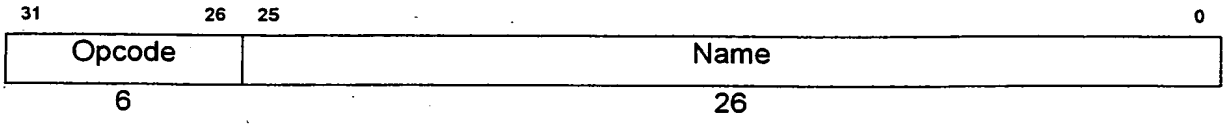

f1 = 0 => rs1 is addressed in the General Purpose Registers;
f1 = 1 => rs1 is addressed in the Special Registers;

f2 = 0 => rs2 is addressed in the General Purpose Registers;
f2 = 1 => rs2 is addressed in the Special Registers;

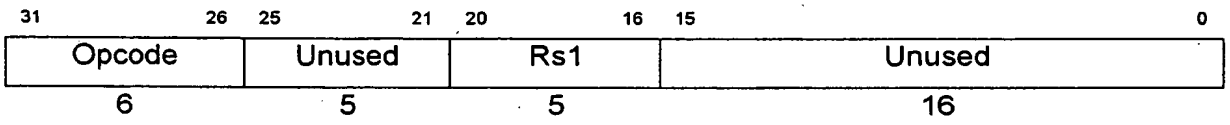
f3 = 0 => rd is addressed in the General Purpose Registers;
f3 = 1 => rd is addressed in the Special Registers.

The third type of instruction is the J-type (jump) instructions which are the unconditional jumps in source code transfer control. There are 4 instructions grouped in this type, Jump, Jump Register, Jump And Link and Jump And Link Register. The two Jump And Link based instructions retain the next instruction address from the jump instruction so that program control can return to the point the jump was executed. This address is stored in the Link Address Register in the Special Register set.

This following is the make up of the Jump and Jump And Link instructions. A 26-bit *name* is sign extended and added to the Program Counter to create the address of the next targeted instruction




The Jump Register and Jump And Link Register instructions are constructed as follows, where rs1 is the General Purpose register address whose contents is the address of the targeted instruction.



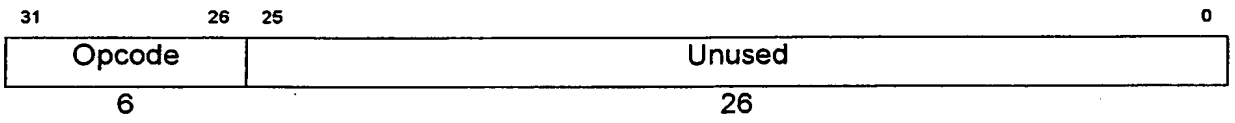

The fourth type of instruction is C-type (control) instruction which is used for processor control type functions. They contain a simple opcode with no register or immediate referenced.

The HALT instruction will stall the EVE Processor pipeline and continued operation will not commence until an interrupt is received.

The RET instruction transfers control back to the section of code jumped from by a JAL or JALR instructions.

The TRAP instruction is a mechanism for allowing software to transfer from the main code to the Exception Service Routine.

The RFE instruction returns control from the Exception Service Routine back to the main code after either a TRAP instruction or an interrupt has been serviced.



In most cases namely R type instructions the use of rs1,rs2 and rd is very clear. For example, it could be

add r5,r4,r3 => rd = r5; rs1 = r4; rs2 = r3

However in the case of I type, namely Load and Store type instructions this is not that clear and thus it is necessary to be aware that:

For a STORE e.g. SW offset(R10), R3 rd = R3, rs1 = R10, immediate = offset
For a LOAD e.g. LW R3, offset(R10) rd = R3, rs1 = R10, immediate = offset

Also for Jumps and Branches based on register values( BEQZ, BNEZ, JR, JALR) the register is in rs1 not rd i.e. in bits 21:16 of the instruction word which means rd (bits 25:21) should be zero.

There are three situations that will require the whole pipeline to stop, one of which, namely, the Halt Instruction, has already been discussed. The other two situations are when either the data or instruction memories are not ready. The signals dataRdy and instRdy respectively are asserted in order to indicate to the stall controller that a stall is to occur. They tell the processor if memory accesses have happened or are about to happen. If the memory access does not happen, they stall the processor.

As mentioned already, it is envisaged that many more instructions may be introduced and the processor according to the invention is configured to adapt to such further instructions as defined as no instruction has a fixed opcode.
One set of instructions implement a branch conditioned on the comparison between the selected byte, halfword or word specified in a register and the corresponding byte, halfword or word specified by the immediate (only for bytes) or in another register. A comparison unit that performs all compares can be fully parameterised allowing any datapath size comparisons. A sub-block, of the comparison unit, of two muxes and XOR gates are parameterised to accept data from 1 up to 8-bits. Depending then on the datapath size, any number of these sub-blocks can be instantiated to form the datapath

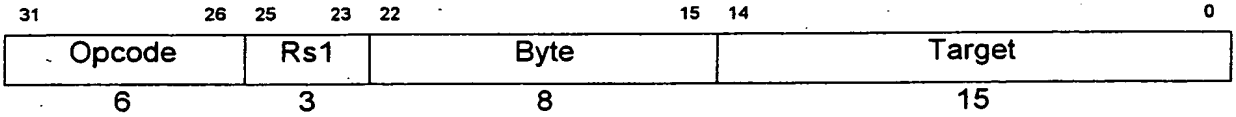
width. A final sub-block, which takes its input from the previous sub-block, which compares this input with zero, will be of datapath size n. Of course, byte comparisons are only allowed when the datapath size is bigger than or equal to 8. When performing a byte comparison, the rest of the sub-blocks will force the output of the XOR gates so that the bits not being tested will not affect the final comparison. These instructions are implemented in two different formats one for the byte immediate branches and the branch on register compare.

These yield in 22 new branch instructions

:**Table 1 – Branch Instructions**

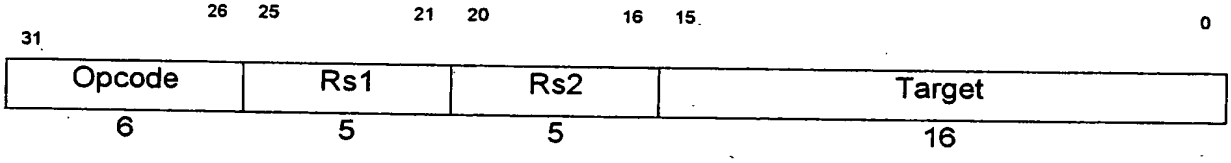
| No. of Opcodes | Mnemonic | Instruction meaning |
|---|---|---|
| 4 | BEQBxI | Branch if byte x equal to byte immediate |
| 4 | BNEBxI | Branch if byte x not equal to byte immediate |
| 4 | BEQBx | Branch if byte x equal to byte register |
| 4 | BNEBx | Branch if byte x not equal to byte register |
| 1 | BEQHU | Branch if upper half equal to half register |
| 1 | BNEHU | Branch if upper half not equal to half register |
| 1 | BEQHL | Branch if lower half equal to half register |
| 1 | BNEHL | Branch if lower half not equal to half register |
| 1 | BEQW | Branch if word equal to register |
| 1 | BNEW | Branch if word not equal to register |

The format of the BEQBxI and BNEBxI instructions is as follows:



When these instructions are executed, the immediate byte in the instruction is compared to a byte in the data item stored in a register pointed to by Rs1. This register address field is only 3 bits in size, therefore, the byte can only be compared to the contents of one of the first 8 GPRs. If the comparison is TRUE, the 15 bit Target is added to the contents of the PC and used as the next address.

The format of the remainder of the Branch on value compare instructions is as follows:




Here, the values contained with in the two registers, addressed by the instruction, are compared and if this compare is true, the Target is added to the current PC and used as the next address.

ALU instructions such as the Adds and Shifts can be implemented to use a carry set by the execution of the previous instruction to affect the carry. Although these are not fully specified, the capacity to implement them is available.

For the Add operation in the ALU, the previous carry is added along with the two source data.

For Shift operations, the previous carry is shift in to the end of the data being shifted and the bit falling of the end is stored as the next carry bit.

The carry bit can also be used to branch on. In executing this instruction, the branch will be taken if the carry is Set or Clear depending on the type of test specified.

As already mentioned, means are provided in the processor to perform logical bitwise operations, such as AND, OR and Exclusive OR on different halves of operands. These

operations can be performed as either I-Type Instructions or R-Type instructions and so adopt the same instruction formats.

After the definition of the pipeline stages elements, the next step is defining the control of their functionality. Because the instructions decoded in the Decode stage are effectively executed in the Execution or Memory stages, some control signals have to be generated and adequately delayed to make them effective at the right time.

The solution adopted by the present invention architecture is sending through the pipeline the control signals along with the data, so they automatically will appear at the right clock cycle in the expected stage. The problems that arise using this configuration, like hazards and stalls, will be discussed below.

Referring to Figs. 2 and 3 all the control signals are generated in the Decode stage 20 depending on the instruction being decoded. They are sent through the pipeline in the case of being used in Execution 30, Memory 40 or Write Back 50 stages or directly used in the Fetch 10 or Decode 20 stages without being registered.

There is only one control signal used in the Fetch stage 10. It is the select signal for the PC mux. The decision is taken in the Decode stage 20 after deciding if the PC should be just incremented by four (the normal program flow) or be loaded with a different value.

The Decode stage 20 not only generates the control signals for other stages but also generates control signals for itself. It is the case of the signal controlling the sign extend unit. The immediate value coming in the instruction is either sign extended or padded with zeroes, depending on the operation being signed or unsigned. The select signal indicates which extension has to be done.

The Execution stage 30 also requires control signals for the muxes choosing the proper source operands for the ALU 31 operation and the ALU 31 needs a signal indicating which operation has to perform on them. There are also four control signals passing through this stage. They will be used in the following stages.

Two of the four control signals received by the Memory stage 40 are used on it. One enabling the data memory 41, thus indicating a load or a store instruction and the other with additional information to generate more control signals for the data memory 41. These two signals are processed in a sub-block in order to generate the RW and size signals to accordingly control the data memory operation. This block also generates a select signal for the data mux. The mux chooses the memory contents in case of performing a load instruction, otherwise the ALU 31 result is passed to the final stage. The other two control signals pass through this stage and will be used in the last one.

Despite the Write Back stage 50 does not have hardware at all, it passes the data to be written, as well as the destination register address and the RW signal to the general purpose and special registers 22.

The flow of any instruction being executed in the processor starts in the Fetch stage, where PC addresses instruction memory 11 and the instruction is read from that position. In the next clock edge, it is registered to the Decode stage 20 where the main decisions are taken. As the result of those decisions, the proper control signals are generated and sent to allow its execution.

In the case of an R-type instruction, the first action done in the Decode stage 20 is addressing the source registers 22. The content of these registers is registered out to the Execution stage 30 along with the ALU 31 operation, the select signals for the source operand muxes, the destination register address and the write enable signal. The rest of the control signals are driven to default. The select signal for the PC mux will chose PC+4 due the program flow will normally continue.

In the Execution stage 30, the source operand muxes pass the corresponding source operands and the operation indicated by the ALU operation signal is performed on them. The result is registered to the Memory stage. This stage and the Write Back stage 50 pass the ALU result along with the address and control signals to the registers 22 to be written. It is because there is no data memory access to perform.

When performing an I-type instruction involving an ALU 31 operation, the situation is similar to the one described above, except that the second source operand is an

immediate. It is extended and registered in the Decode stage 20. In the Execution stage 30 it is chosen as a second operand, instead of Rs2, by means of the select signal. If the instruction is a load or a store, Rs1 and the immediate are added to form the data memory 41 address. In the case of a store, Rs2 holds the data to be stored, so it is also passed to the Memory stage 40 and there is no destination register.

The control signals indicating the type of load or store instruction (signed or unsigned, byte, halfword or word) sent from the Decode stage 20 are processed in the Memory stage 40 to produce the proper control signals for the memory. It also generates a select signal for the mux choosing between the memory data (in case of a load instruction) or the ALU 31 result.

The store instruction is finished in the Memory stage 40, because there is no data to write back to the registers 22. Nevertheless, the Write Back stage 50 sends the data, destination register address and write enable signals to the registers 22 as usual. In this case, the data sent is the ALU 31 result and the destination register is R0 (not writable).

When the I-type instruction is a Branch the actions taken are different. The register 22 indicated in the instruction is addressed as usual, but its content is compared with zero to decide whether the branch should be taken or not. It is done in the Decode stage 20. Depending on that decision, next PC is selected adequately in the Fetch stage. The optional Brach Instructions which performs the comparison between selected byte, halfword or word specified by the immediate or in another register, are also done in the Fetch stage 20.

The control signals sent through the pipeline are defaulted because any actions are required further on. The Execution stage 30 thus performs an addition on the register 22 addressed and R0 and the destination address is set to R0. It is equivalent to perform a NOP which is defined as: ADD R0,R0,R0.

In the case of a branch taken, a signal is set in the Decode stage 20 to indicate that the next instruction has to be annulled. It is because that instruction was fetched while decoding the branch instruction but it should not be executed. If the branch is not taken, the program flow normally continues.

To annul an instruction means that despite it has been fetched, it will not be executed. To do so, the Decode stage 20 sends to the pipeline a NOP, ignoring the contents of the instruction.

The J-type instructions change the value of next PC unconditionally. What is decided in the Decode stage 20 is which value of next PC has to be chosen and whether to store or not the value of actual PC in order to continue the normal program flow after returning from the jump routine. These instructions cause the next instruction to be annulled.

The J instruction includes an offset to be added to the actual PC to form the target address. That address is chosen by the select signal of the PC mux as the value for next PC. A NOP is sent to the pipeline because nothing has to be calculated in the Execution stage onwards.

JAL instruction does the same, except that actual PC is stored in the Link Address Register. When the RET instruction is found, the value stored in LAR is loaded into next PC to allow the program flow to continue.

JR and JALR address a register, which content is directly loaded as next PC. Again, in the case of a JR a NOP is sent to the pipeline and in the case of a JALR, the value of actual PC is stored in LAR and when the RET instruction is found, the value stored in LAR is loaded back into next PC to allow the program flow to continue.

The control transfer instructions accordingly change the value of PC. The instruction TRAP or an interrupt cause next PC to be loaded with a predetermined address and actual PC to be stored in Exception Address Register (EAR). The content of that address is either the first instruction of the Exception Service Routine (ESR) or an instruction to jump to it. RFE marks the end of the ESR and causes next PC to be loaded with the contents of EAR.

RET does the same as RFE, but loading the content of LAR instead. The instruction HALT causes the whole pipeline to stall until an interrupt is received. Every stage keeps

doing the actions they were doing when the HALT instruction came in, until the pipeline is released and the inputs of the stages are able to change.

Fig. 4 illustrates the implementation of a bitwise logic X instruction to be carried out on two operands, A and B, each having an even number of bits. First of all, the upper and lower halves of data of the first operand are swapped before the bitwise logical operation is carried out producing the result. The bitwise logical operation block may represent any of AND, NAND, OR, NOR, XOR or XNOR operations. The instructions produced are ANDX, NANDX, ORX, NORX, XORX and XNORX.

For example, if we take two inputs of four bits each, input A being 1001 and input B being 0011, and we are to perform an ANDX operation on the data, then we would first of all have to swap the data in the top half of input A with the data in the bottom half of input A, namely, bits 2 and 3 with bits 0 and 1. This would mean input A would become 0110. The operands are then passed through an AND gate giving a result of 0010.

Another similar type of instruction is one bitwise logic S instruction, as shown in Fig. 5. In this case of bitwise logic S instructions, the logic operation, i.e. AND, NAND, OR, NOR, XOR, XNOR is performed before the data in the upper half of the result is swapped with the data in the lower half of the result. These instructions are denoted by ANDS, NANDS, ORS, NORS, XORS and XNORS.

When the X and S operations are performed on data of uneven bit number, one bit of data is discarded and the remainder of the data is operated upon in the manner already described. It has been found convenient to discard the central bit of data in data of uneven bit.

All of the above-mentioned functions may be achieved using extensive cross-wiring techniques.

More detail with the addition of the extra hardware to prevent stalls and hazards, which includes the muxes in front of the registers and the forwards, the pipeline shown in Fig. 3 becomes the one shown in Fig. 6.

Fig. 7 illustrates the top level break down of the core into 4 main areas.

There are shown four essential areas namely the Instruction Unit 61, the register unit 62, the execution unit 63 and the data unit 64.

The instruction unit 61 is the section where all work is done with the instruction, control of fetching it from the instruction memory, decoding it and setting control signals for the rest of the processors.

The register unit 62 is separated from the Instruction Unit. This is aimed towards synthesis, as there will be a large amount of actual registers implemented. It is addressed by the decoding of the instruction to present operands to the Execution Unit.

The execution unit 63 is the implementation of the Execution stage of the EVE Processor pipeline. This is in its own block as concentration can be put on it because of its importance and possibly its critical timing.

Finally, the Data Unit 64 is the remainder of the processor, which in effect does something with data, writes data to memory reads data from memory and then writes data back to a register of the processor.

Data being written to or from memory may be of a different length to the datapath width. Often, the data will have to be extended to populate the entire datapath width. If the data is in signed format, this is particularly important as the sign of the data must be maintained. A signal is generated, *dmSESel*, to select which kind of sign extension has to be performed on the data coming from memory. It is asserted when loading a byte or a halfword, and also depends on the type of load being performed (signed or unsigned). Otherwise, data coming from memory goes through this sub-block without being changed. The encoding values are shown in Table 2.

**Table 2 - Encoding values for dmSESel signal**

| code | Label | Comments |
|---|---|---|
| 0xx | PASSTHROUGH | Pass word through |
| 100 | ZEXTBYTE | Zero Extend byte |
| 101 | SEXTBYTE | Sign Extend byte |
| 110 | ZEXTHALF | Zero Extend halfword |
| 111 | SEXTHALF | Sign Extend halfword |

The data memory Sign Extend Unit not only performs the sign extension indicated by *dmSESel*, but also places the correct byte or halfword coming from data memory into the register. That operation depends on the signal *endian*, which indicates if the system is accessing data memory in little endian mode (*endian* = 1) or big endian mode (*endian* = 0). The flow diagram for the block is shown in Fig. 8, where a 32 bit data path is assumed.

Once the size (byte or halfword) is determined, the signal endian is checked in order to pick the correct part of the data and place it in the register. Then, the corresponding byte or halfword is either sign extended or zero extended. If the size indicates word, the content of the memory position addressed is just placed in the register, as it passes through this module without suffering any transformations.

The Register unit 62 is built up of 3 sections. Fig. 9 shows an overview of the Register Unit 62 and its main components. The Register Files themselves are separated into two banks, the General Purpose Registers and the Special Registers. All registers are synchronous to the system clock *sysClk*.

Within the General Purpose Registers (GPRs) there are 32 addressable registers. There are in fact only 31 registers, each of these being 32-bit wide, with register 0 not being made up of actual registers but is a constant 32-bit zero. A block diagram of the GPRs is shown in Fig. 10, which breaks down into 3 sections.

By means of the parameterisation of the EVE architecture, the GPR block can be outside of the processor. In that case the inputs to the block will be driven to that external block and its outputs will be connected to the corresponding inputs in the Register Muxes block to be chosen as source operands if selected.

The signal *dataBack* returns to the registers in what is the Write Back stage of the processor pipeline. It contains the new data to be written to a register by an instruction.

This demultiplexing is controlled by the *addrDestReg* bus that contains the address of the destination register and the write enable signal, *writeRegEn*. By ANDing this write enable signal with the inverse of bit 5 of the destination register address creates a select for the GPRs. Each of the remaining bits of the destination register address are ANDed with this generated enable signal and if the enable is not set this will cause a write to register 0 which does not store a value.

The Register Block is implemented just as registers. Note that these must maintain their value on every clock period.

When an instruction addresses a register so as to use its contents, it puts a 6-bit address on one of two busses, to set this data up as operand 1 or operand 2 or both. All but the top bits of these two busses drive multiplexers that select the register value, as seen in Fig. 10 above.

The Special Registers allow up to 32 addressable registers, where the first four are always present as they keep specific processor information. These four registers are the Reason Register at binary address 100000, the Interrupt Register at binary address 100001, the Exception Address Register (EAR) at binary address 100010 and the Link Address Register (LAR) at binary address 100011. The bit field definitions of these 4 registers have been described above.

The Exception and Link Address Registers are just written to as normal through the pipeline and does not need any special logic around it. The Reason Register and the Interrupt Register, however, require further logic to handle resets and external interrupts as they happen.

While those four registers are always inside the processor, the rest of the special registers can be either inside or outside it. The placement of the registers is determined by the parameterisation.

In the case of the registers being outside the processor, the outputs of the register bank go to the Special Registers module shown in Fig. 9 above. It is to supply the right addressed Special Register to the Register Muxes.

This register holds two bits of information, the enable bit and the pending bit. So, it must react to the resets and the interrupt control signals. A rising edge must be detected on *extint*, the signal from the external interrupting source. It will set the pending bit. This must be maintained until either an internal acknowledge or a reset has been received. At this point an acknowledge must also be given back to the external interrupting source. The processor can read from this bit, but not write to it.

On the other hand, the enable bit can be read from and written to by the processor. When this bit is set, and incoming interrupt will be serviced, otherwise it will be ignored.

The Reason Register has to show the present state of the processor and cannot wait for the latency of data passing through the pipeline. It has to react to a hardware reset *sysReset*, an illegal instruction *sReset* and either a Trap instruction or an Interrupt being serviced.

The exception address register will keep the PC value at the clock cycle a TRAP or an interrupt changes the program order to be serviced. The instruction corresponding to this PC value is annulled, so its execution has to be restarted once the Exception Routine is finished.

EAR is written by the though the pipeline when a TRAP or an interrupt are detected and can be serviced. The address stored is read by the instruction RFE, which causes next PC to be loaded with it in order to fetch again the instruction previously annulled.

The link address register is used when the execution of JAL or JALR will cause a change in the program order. They change the value of PC to the target address and the annulation of the instruction just been fetched. The address of that instruction is stored in LAR.

LAR is written by the though the pipeline by JAL or JALR. Instruction RET will read the stored address, which will be loaded into next PC to restart the execution of the instruction previously annulled.

There are only two operands required by the Execution Unit and either a General Purpose register or a Special register can be selected at any one time for each of these operands. The data returned to a specific register may be needed in the next clock period before it can be written back into the register, therefore, a forwarded path of *dataBack* is required.

The case of a stall occurring also needs to be covered where the data in the previous clock period needs to be sent through again. The block diagram in Fig. 11 shows the two muxes that perform the selection of the data as the sources for the Execution Unit. The control for these muxes is handled in the Instruction Unit, where the instruction is decoded and thus the decision of what data to use is implemented. The data selected is then registered out of this pipe stage and into the Execution stage.

The inputs regS1 and regS2 are driven by the outputs of the General Purpose Registers, either being placed inside or outside. This fact is reflected in the block instantiation, where the actual signals are connected to the formal signals of the block.

The situation of the signals sRegS1 and sRegS2 is different. They always come from the Special Registers block independently of part of the registers being inside or outside.

In essence, what has been produced is a generic processor, that is, a processor that can be adapted to the requirements of the job in hand. This generic processor is an outline design for a device that may be stored as a computer program on a record medium. In other words, the processor may be seen as a template from which further processors may be derived from. The general design is there and all the designer has to do is to input his specific requirements and generate a specific processor from the template. The designer may then go and realize the designed processor. This may be on a purpose built chip or the designer may realize the processor on a Field Programmable Gate Array (FPGA), depending on his/her own requirements.

Some of the embodiments of the invention described with reference to the drawings comprise processes performed in computer apparatus. The invention also extends to computer programs, particularly computer programs on or in a carrier adapted for putting the invention into practice. The code may be in source code, object code or a code intermediate source and object code or any other form suitable for use in the implementation of the methods according to the invention.

The carrier may comprise a storage medium, for example, a ROM, CD or semiconductor, floppy disk or any other recording medium. Alternatively, the carrier may be a transmissible carrier such as an electrical or optical signal that may be conveyed by an electric or optical cable or any other means. When the program is embodied in a signal on such cables or other means, the carrier may be constituted by such means.

The carrier may also be an integrated circuit in which the program is embedded, the integrated circuit being adapted for performing or for use in the performance of relevant methods.

In the specification the terms "comprise, comprises, comprised and comprising" or any variation thereof and the terms "include, includes, included and including" or any variation thereof are considered to be totally interchangeable and they should all be afforded the widest possible interpretation and vice versa.

The invention is not limited to the embodiment hereinbefore described, but may be varied in both construction and detail within the scope of the claims.

CLAIMS

1. A processor (1) having a number of components including at least:

a configurable arithmetic and logic unit (4);

a plurality of registers (3);

memory access; and

datapaths (5) between the components,

characterised in that:

the datapath width is of variable bit size namely n bits;

the number of the components are selectable;

where appropriate the components are of n bit size; and

each component is configured to handle data having one of two sizes

$\leq n$ or $> n$.

2. A processor (1) as claimed in claim 1 comprising:

means to select the number A and size of each component;

means to select the datapath width;

means to configure the components for that datapath width; and

means to compare the width of a data input to the selected datapath width that has been chosen for the component.

3. A processor (1) as claimed in claim 1 or 2 in which the immediate data of an instruction occupies a fixed size in memory and when the size is greater than n bits, the size is truncated to n bits.

4. A processor (1) as claimed in claim 1 or 2 in which the immediate data of an instruction is limited in size to a preset number of bits and when this number is less than n the immediate data is extended to n bits wide.

5. A processor (1) as claimed in claim 4 in which there is provided means to determine whether the immediate data of an instruction is in signed or unsigned format.

6. A processor (1) as claimed in claim 5 in which on determining that the immediate data of an instruction is in signed format the immediate data is expanded to n bits wide with the vacant bits being populated with the most significant bit (MSB) of the immediate data.

7. A processor (1) as claimed in claim 5 in which on determining that the immediate data of an instruction is in unsigned format the immediate data is expanded to n bits wide with the vacant bits being populated by zeros.

8. A processor (1) as claimed in any preceding claim in which the processor has special purpose registers and general purpose registers (22).

9. A processor (1) as claimed in claim 8 in which the general purpose registers are mounted external of the processor (1).

10. A processor (1) as claimed in claim 8 or 9 in which one or more of the special purpose registers are mounted external of the processor (1).

11. A processor (1) as claimed in any of claims 8 to 10 in which the registers are configured to allow their content to be written to memory external of the processor (1).

12. A processor (1) as claimed in any of claims 8 to 11 in which the special registers may be written to external memory and used as general registers.

13. A processor (1) as claimed in any of claims 8 to 12 in which all the general registers are n bits wide.

14. A processor (1) as claimed in claim 13 in which the most significant bit (MSB) of a location in a register is the $(x-1)^{th}$ bit and the MSB of data of size n is the $(n-1)^{th}$ bit.

15. A processor (1) as claimed in claim 14 in which means are provided for determining whether the n bit data is in signed or unsigned format.

16. A processor (1) as claimed in claim 15 in which the means are provided for writing the n bit data to an address of size x bits greater than n bits, the bits in positions between (x-1) and (n-1) inclusive are populated by the MSB of the n bit when the n bit data is in signed format.

17. A processor (1) as claimed in claim 15 in which means are provided for writing the n bit data to an address of size x bits greater than n bits, the bits in positions between (x-1) and n are populated by zeros when the n bit data is in unsigned format.

18. A processor (1) as claimed in claim 15 in which means are provided for writing the n bit data to an address of size x bits less than n bits, the bits in positions between (n-1) and x inclusive are truncated.

19. A processor (1) as claimed in any preceding claim in which when it is required to perform logical operations on data in the high order address of one word with data in the lower order address of another word when n is an even number, means are provided to perform logical operations on the data in the top half of the word with data in the bottom half of the other word.

20. A processor (1) as claimed in any of claims 1 to 19 in which when it is required to swap the upper order address of a word with the lower order address of that word

when x is an even number, means are provided to swap the top half of a word with data in the bottom half of the word.

21. A processor (1) as claimed in any of claims 1 to 20 in which when it is required to perform logical operations on data of one word with data of another word and subsequently to swap the data in the upper half of the result with data in the lower half of the result and when x is an even number, means are provided to swap the data in the upper half of the result with the data in the lower half of the result.

22. A processor (1) as claimed in any of claims 1 to 21 in which when it is required to perform logical operations on data in the high order address of one word with the data in the lower order address of another word when n is an uneven number means are provided to discard the central bit of each word and to perform logical operations on the data in the top half of the word with data in the bottom half of the other word.

23. A processor (1) as claimed in any of claims 1 to 22 in which when it is required to swap the upper order address of a word with the lower order address of that word where n is an uneven number, means are provided to discard the central bit of the word and to swap the upper order address of the word with the lower order address of that word.

24. A processor (1) as claimed in any of claims 1 to 23 in which when it is required to perform logical operations on two words of data and to subsequently swap the data in the high order address of the result with the data in the low order address of the result when x is an uneven number, means are provided to discard the central bit and swap the data in the high order address with the data in the low order address of the result.

25. A processor (1) as claimed in claims 19 to 24 in which the means to perform the swapping of data is provided by cross-wiring techniques.

26. A processor (1) as claimed in any preceding claim in which there is provided additional logic circuitry for specific logical operations and an interface for communication with the additional logic circuitry.

27. A processor (1) as claimed in claim 26 in which additional logic circuitry may be added subsequent to the realisation of the processor.

28. A processor (1) as claimed in any preceding claim in which there are at least two processors sharing common general purpose registers.

29. A processor (1) as claimed in any preceding claim in which there are at least two processors sharing common special purpose registers.

30. A processor (1) as claimed in any preceding claim in which the processor is embodied in a software program.

31. A processor (1) as claimed in claim 30 in which the processor embodied in a software program is stored on a record medium.

32. A processor (1) as claimed in claim 30 in which the processor embodied in a software program is carried on an electrical carrier signal.

33. A processor (1) having the structure of a processor as claimed in any preceding claim in which n is given a desired value.

34. A processor (1) as claimed in claim 33 in which A is given a desired value.

35. A processor (1) as claimed in claim 33 or 34 in which processor is embodied in a software program.

36. A processor (1) as claimed in claim 35 in which the processor embodied in a software program is stored on a record medium.

37. A processor (1) as claimed in claim 35 in which the processor embodied in a software program is carried on an electrical carrier signal.

38. A method of designing a generic processor comprising the steps of:

preparing an outline processor in general architecture having a series of components described by blocks or the like interconnected by various datapaths having at least a configurable or arithmetic and logic unit (4), a plurality of registers (3), memory access and such other units and components as are required for a processor of the type being designed and then defining the datapath width of variable bit size, namely n bits,

choosing A components where A is any number that could be chosen;

defining the component size as n bit size; and

programming each component to handle data having one of two sizes, namely

$\leq n$ or $> n$.

39. A method of designing a processor as claimed in claim 38 in which the designer chooses an arbitrary number of components greater than that that would ever be required.

40. A method of designing a customised processor using the processor of claim 2 comprising the steps of:

selecting the datapath width;

selecting the number and size of each component;

configuring the components for that datapath width; and

configuring each component to handle data having one of two sizes, namely

$\leq n$ or $> n$.

41. A computer program comprising program instructions for causing a computer to perform the method of claim 40.

42. A computer program according to claim 41 embodied on a record medium.

43. A computer program according to claim 41 stored in a computer memory.

44. A computer program according to claim 41 embodied in a read-only memory.

45. A computer program according to claim 41 carried on an electrical carrier signal.

46. A computer program according to claim 41 carried on an optical carrier signal.

## ABSTRACT

## "A Data Processor"

A processor (1) of the type having a number of components including at least a configurable arithmetic and logic unit (4), a plurality of registers (3), memory access and datapaths (5), between the components. The datapath width is of variable bit size, namely, n bits, the number of components are selectable and where appropriate, are of n bit size and each component is configured to handle data having one of two sizes namely ≤ n or > n. In essence, there is provided a generic processor that may be tailored to suit the specific tasks, space and computational requirements that have been determined by a designer. A method is also provided for designing such a processor.

cj\spec10-2000\s0432dec00

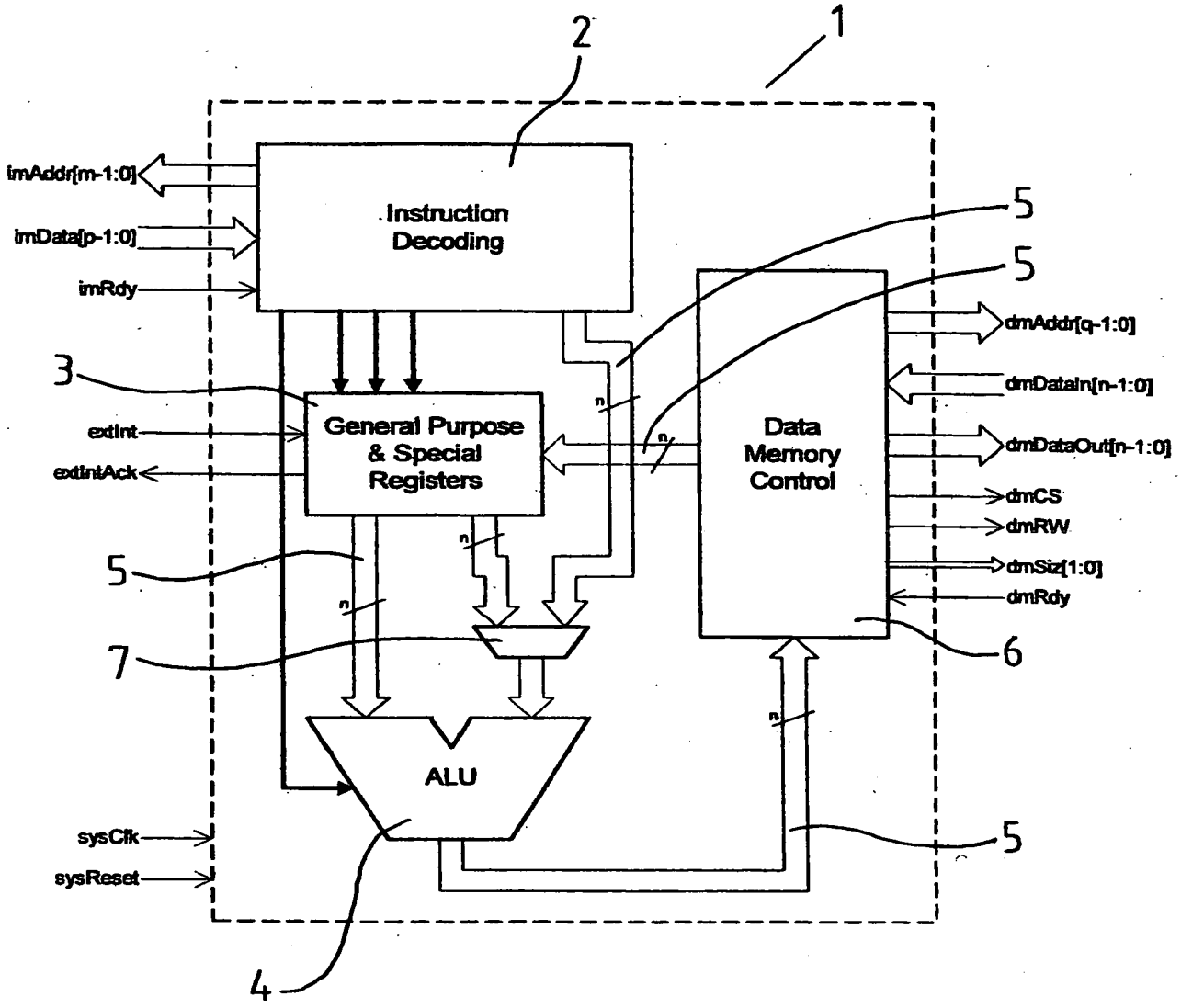

1
2
Instruction Decoding
imAddr[m-1:0]
imData[p-1:0]
imRdy
3
General Purpose & Special Registers
extInt
extIntAck
5
7
ALU
sysClk
sysReset
4
Data Memory Control
dmAddr[q-1:0]
dmDataIn[n-1:0]
dmDataOut[n-1:0]
dmCS
dmRW
dmSiz[1:0]
dmRdy
6
n

Fig. 1

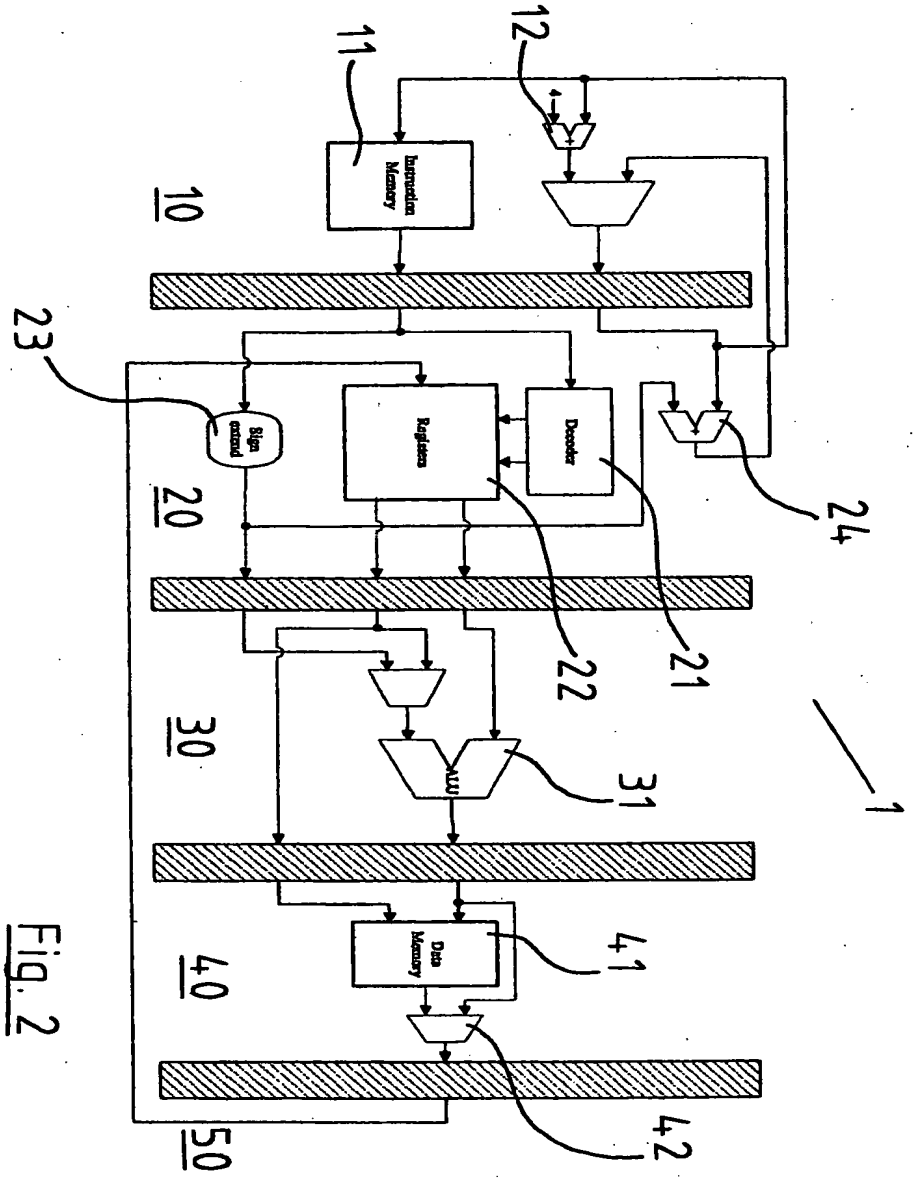

11
12
Instruction Memory
10
23
Sign extend
Registers
Decoder
24
20
22
21
30
ALU
31
1
41
Data Memory
40
42
50

Fig. 2

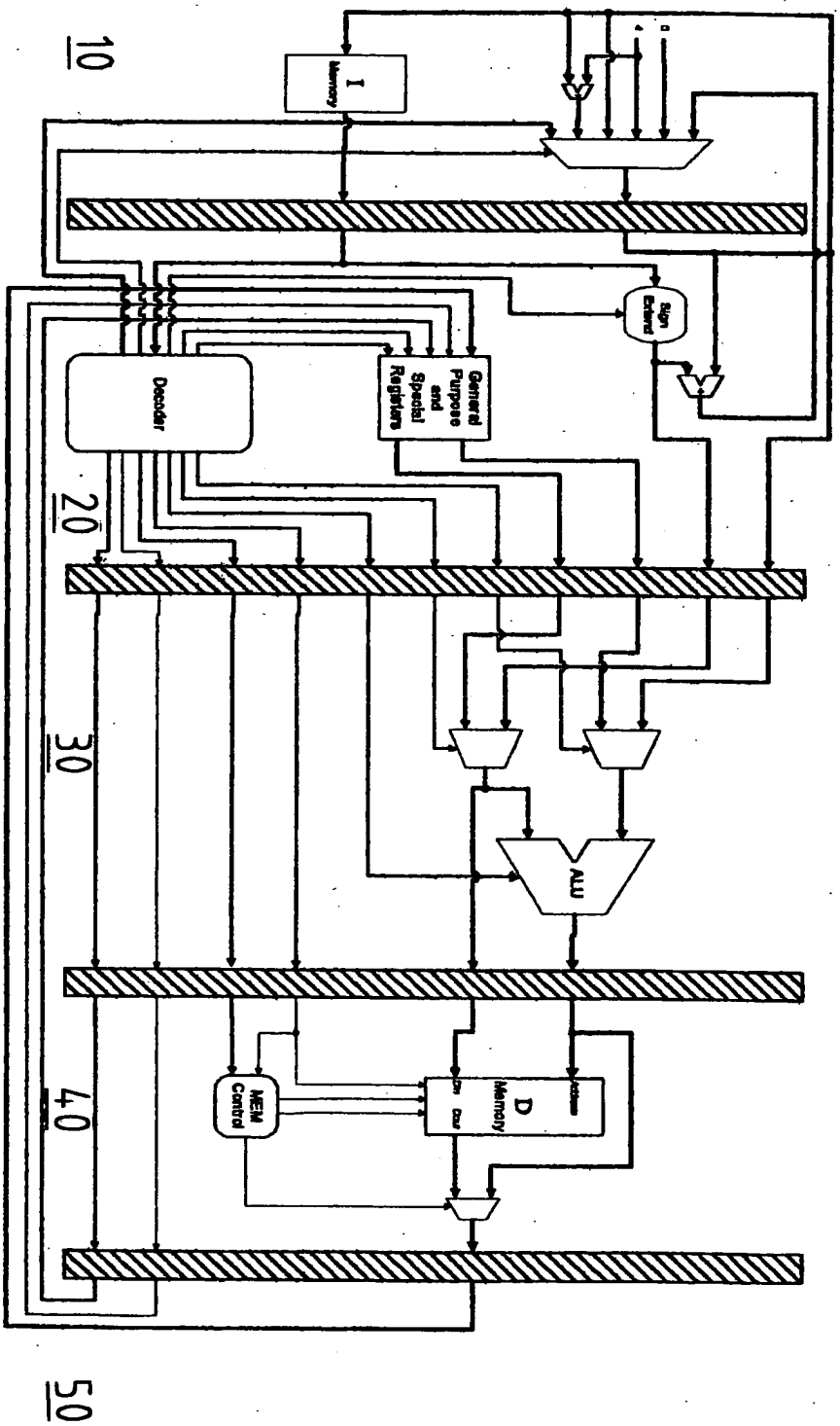

10
I Memory
Sign Extend
Decoder
General Purpose and Special Registers
20
30
ALU
MEM Control
D Memory
40
50

Fig. 3

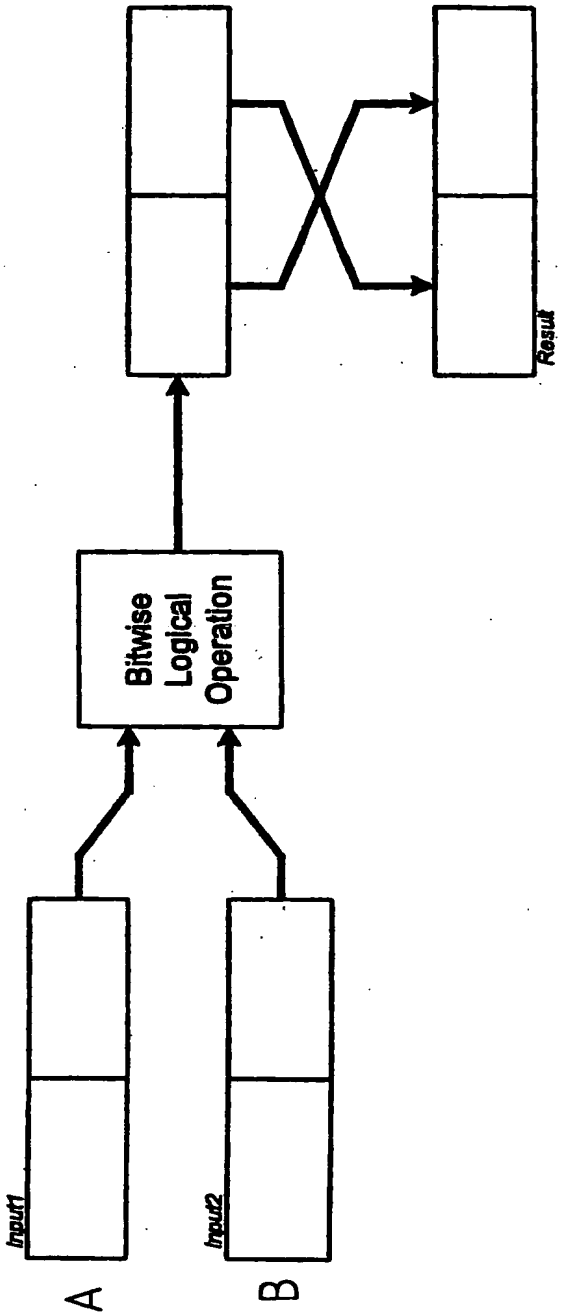

Input1
Input2
A
B
Bitwise Logical Operation
Result

Fig. 5

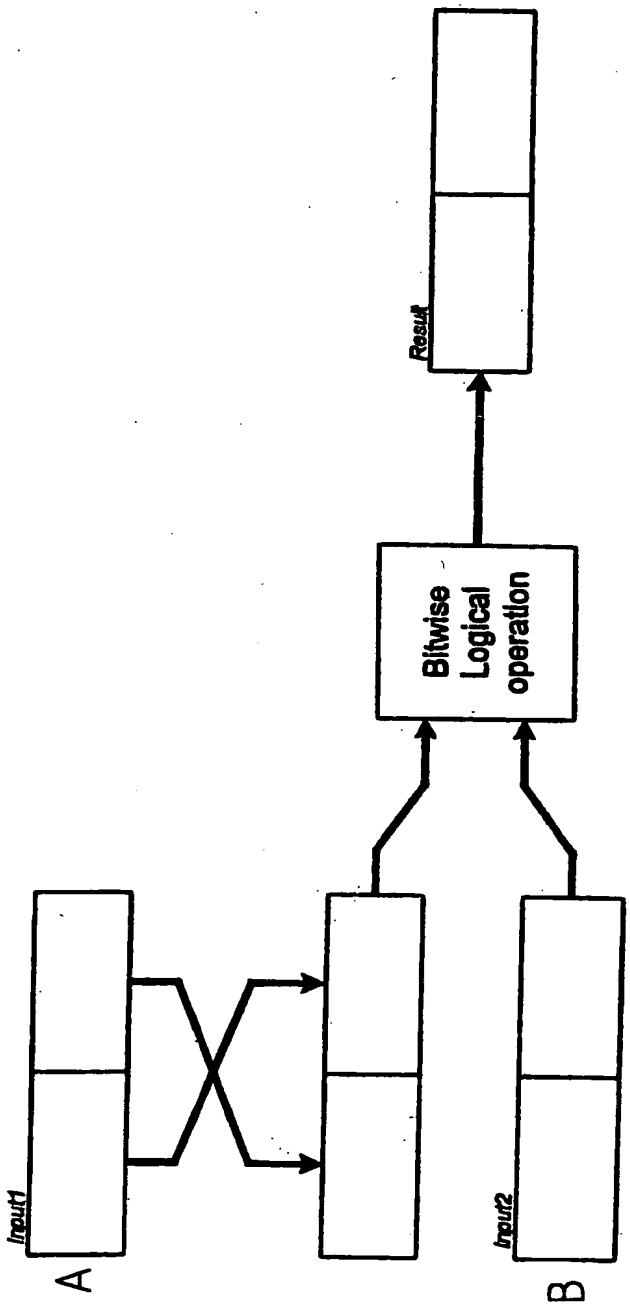

Input1
Input2
A
B
Bitwise Logical operation
Result

Fig. 4

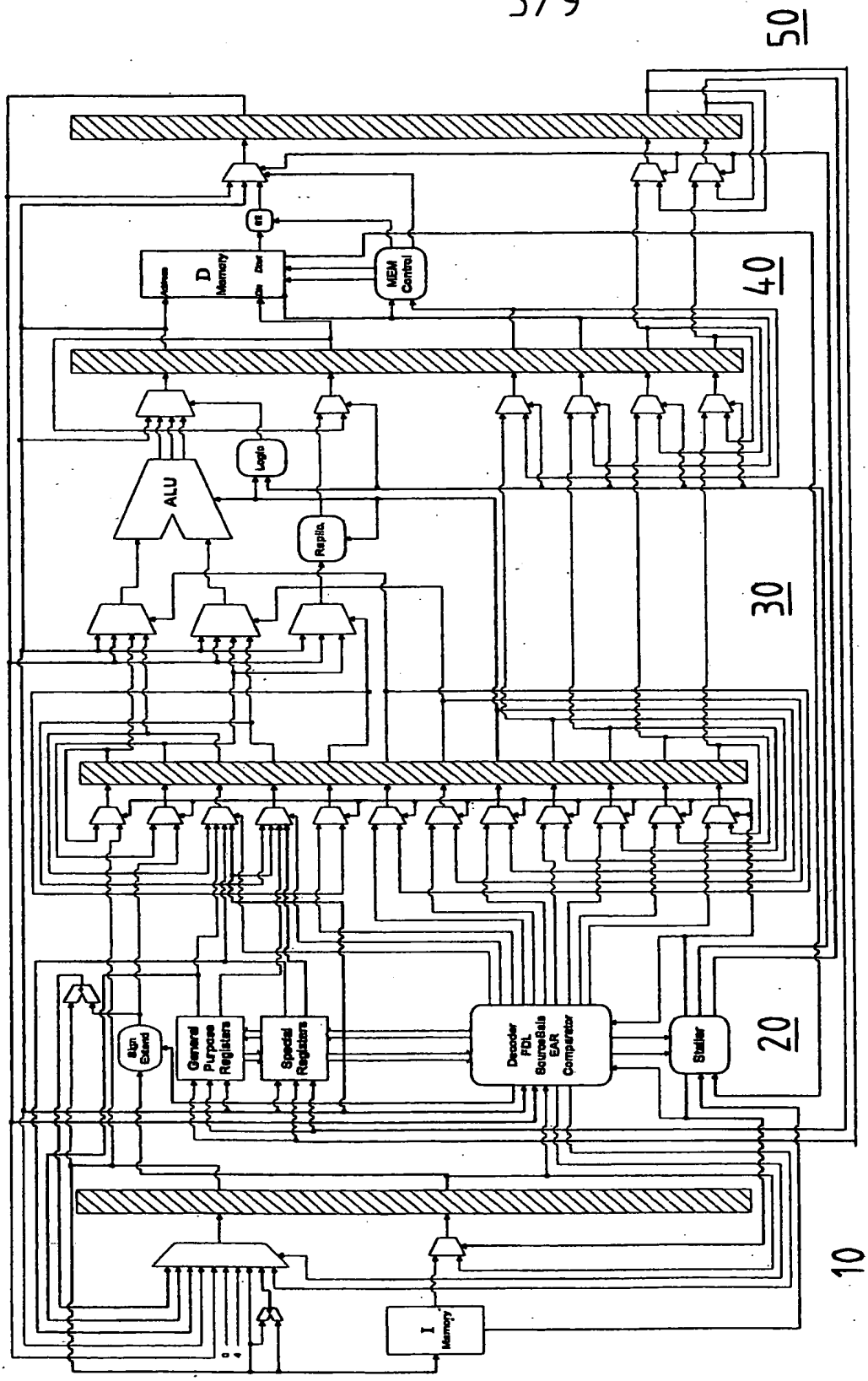

50
40
30
20
10
D Memory
MEM Control
Logic
ALU
Replic.
Sign Extend
General Purpose Registers
Special Registers
Decoder PDL SourceBala EAR Comparator
Staller
I Memory

Fig. 6

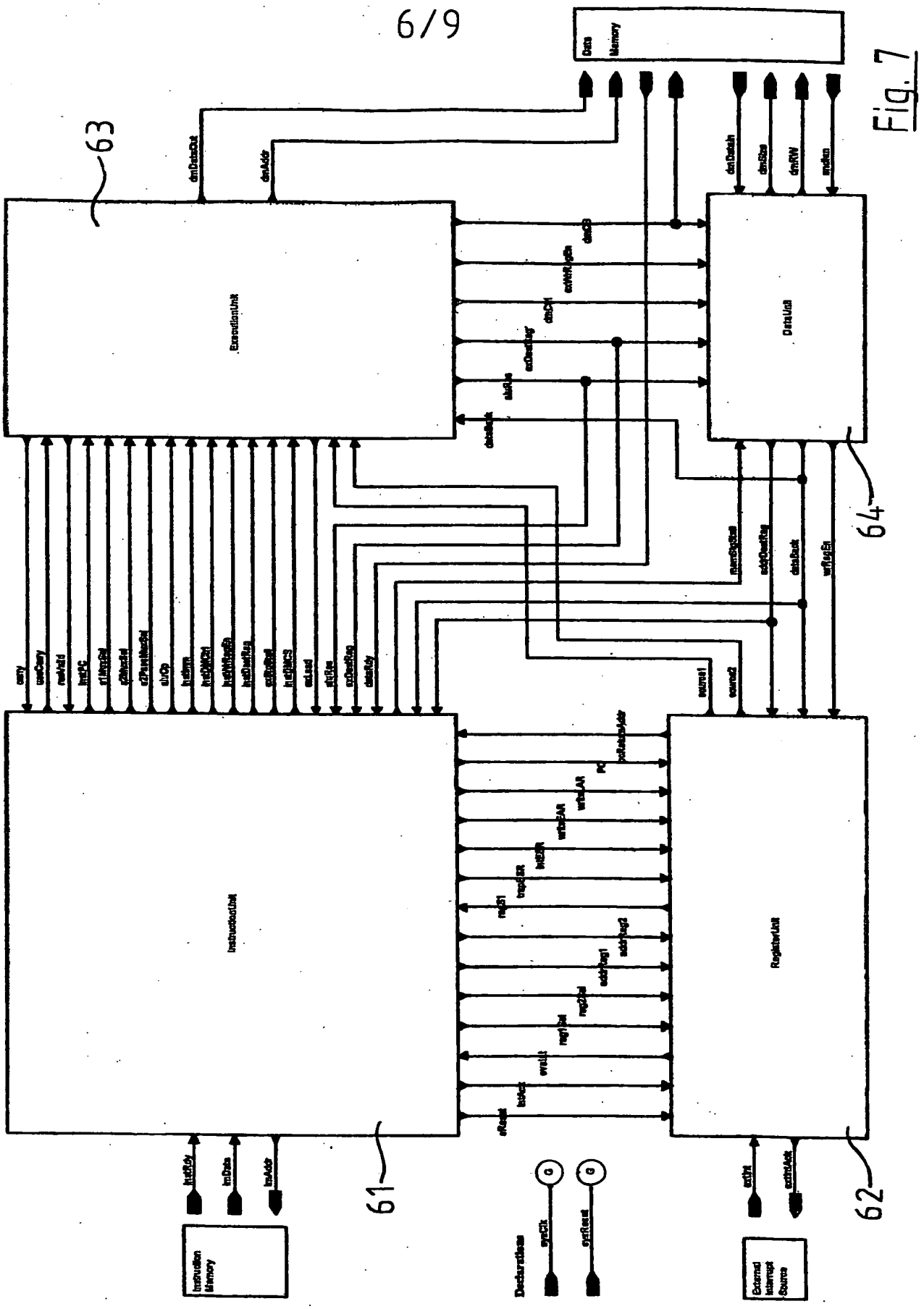

Data Memory
63
ExecutionUnit
DataUnit
64
InstructionUnit
RegisterUnit
61
62
Instruction Memory
External Interrupt Source
Declarations

Fig. 7

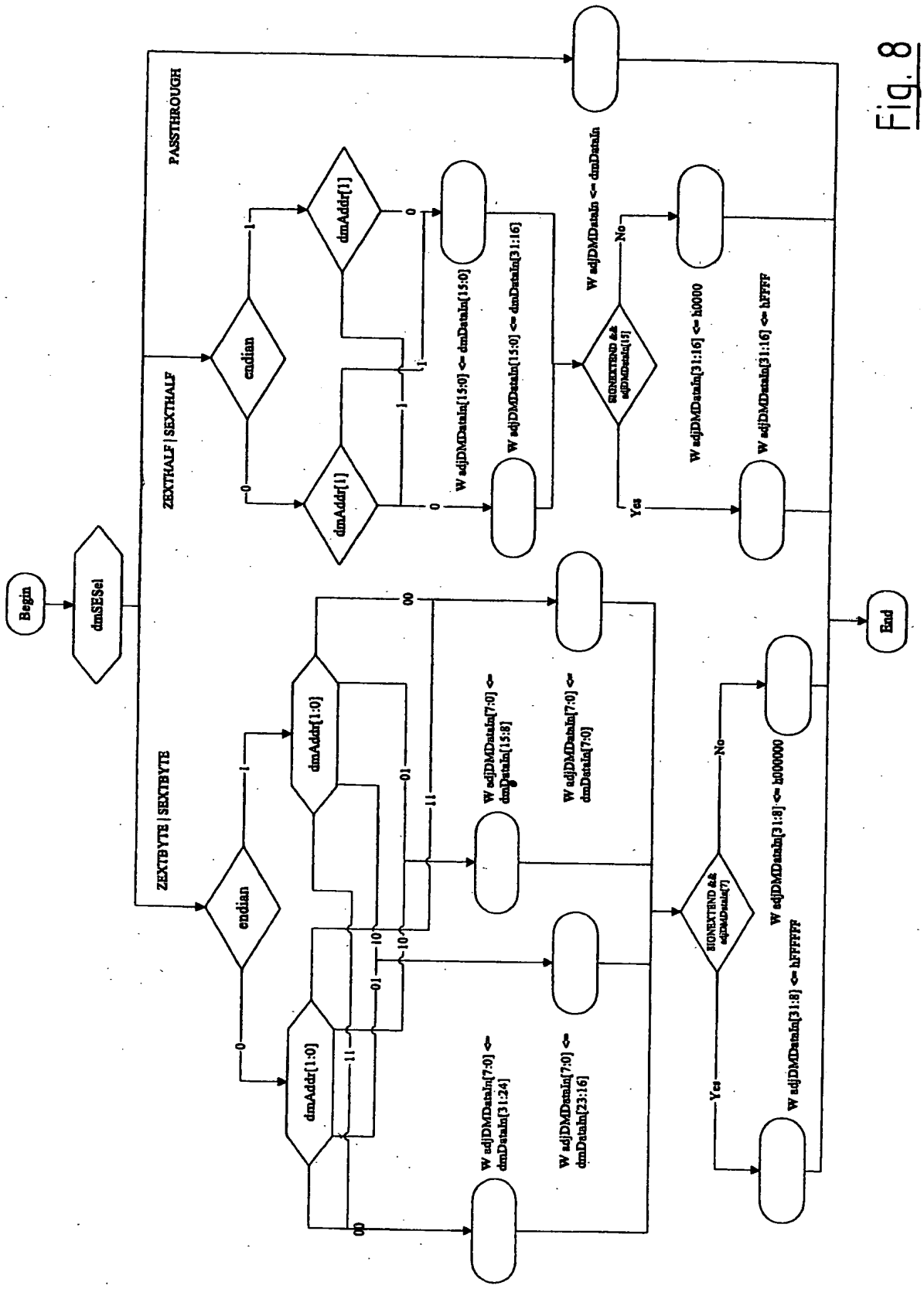
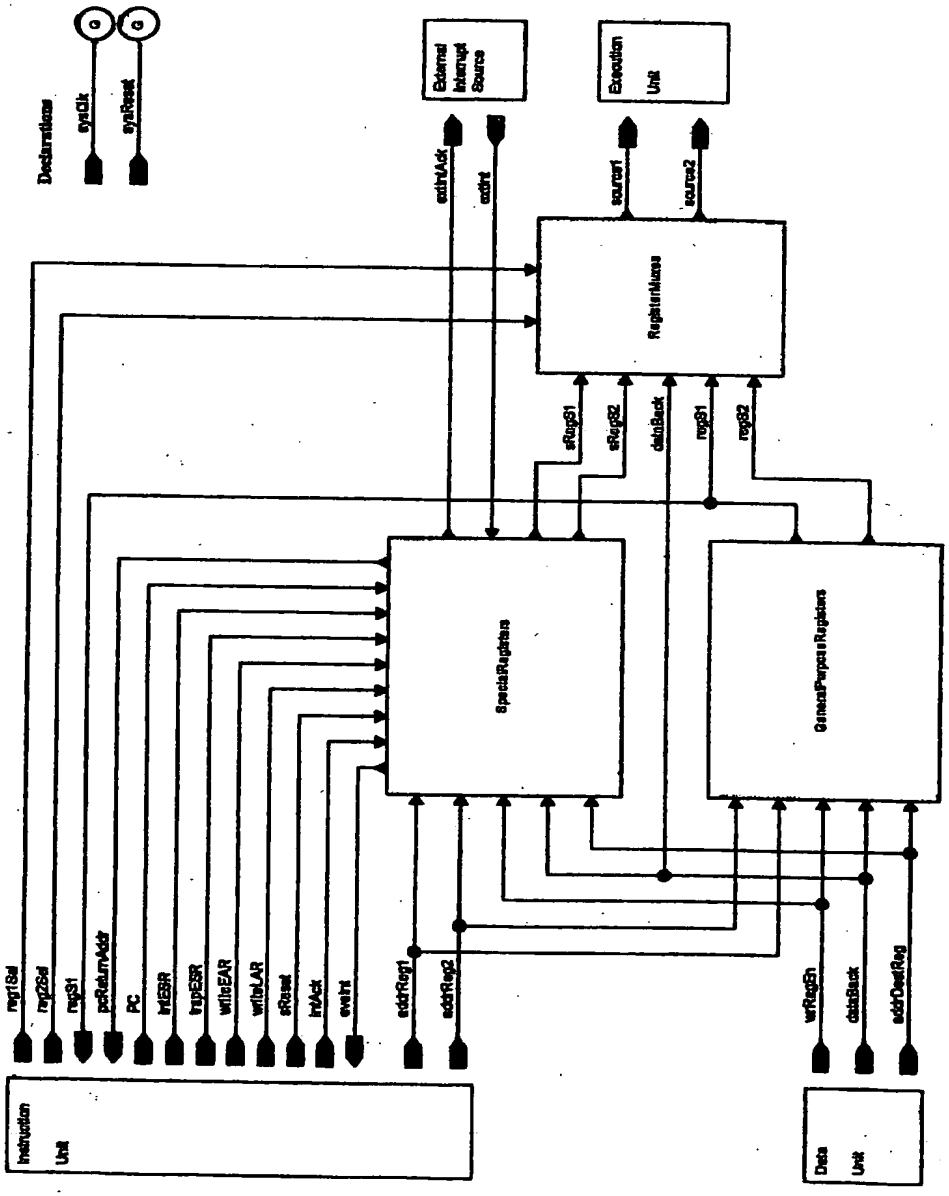

Begin
dmSESel
PASSTHROUGH
ZEXTHALF | SEXTHALF
ZEXTBYTE | SEXTBYTE
endian
dmAddr[1]
dmAddr[1:0]
W adjDMDatain <= dmDatain
W adjDMDatain[15:0] <= dmDatain[15:0]
W adjDMDatain[15:0] <= dmDatain[31:16]
SIGNEXTEND && adjDMDatain[15]
W adjDMDatain[31:16] <= h0000
W adjDMDatain[31:16] <= hFFFF
W adjDMDatain[7:0] <= dmDatain[15:8]
W adjDMDatain[7:0] <= dmDatain[7:0]
W adjDMDatain[7:0] <= dmDatain[31:24]
W adjDMDatain[7:0] <= dmDatain[23:16]
SIGNEXTEND && adjDMDatain[7]
W adjDMDatain[31:8] <= h000000
W adjDMDatain[31:8] <= hFFFFFF
No
Yes
End

Fig. 8

Fig. 9


Declarations
sysClk
sysReset
External Interrupt Source
Execution Unit
extIntAck
extInt
source1
source2
RegisterMuxes
sRegS1
sRegS2
dataBack
regS1
regS2
SpecialRegisters
GeneralPurposeRegisters
regISel
regZSel
regS1
pcReturnAddr
PC
intESR
trapESR
writeEAR
writeLAR
sReset
intAck
exeInt
addrReg1
addrReg2
wrRegEn
dataBack
addrDestReg
Instruction Unit
Data Unit

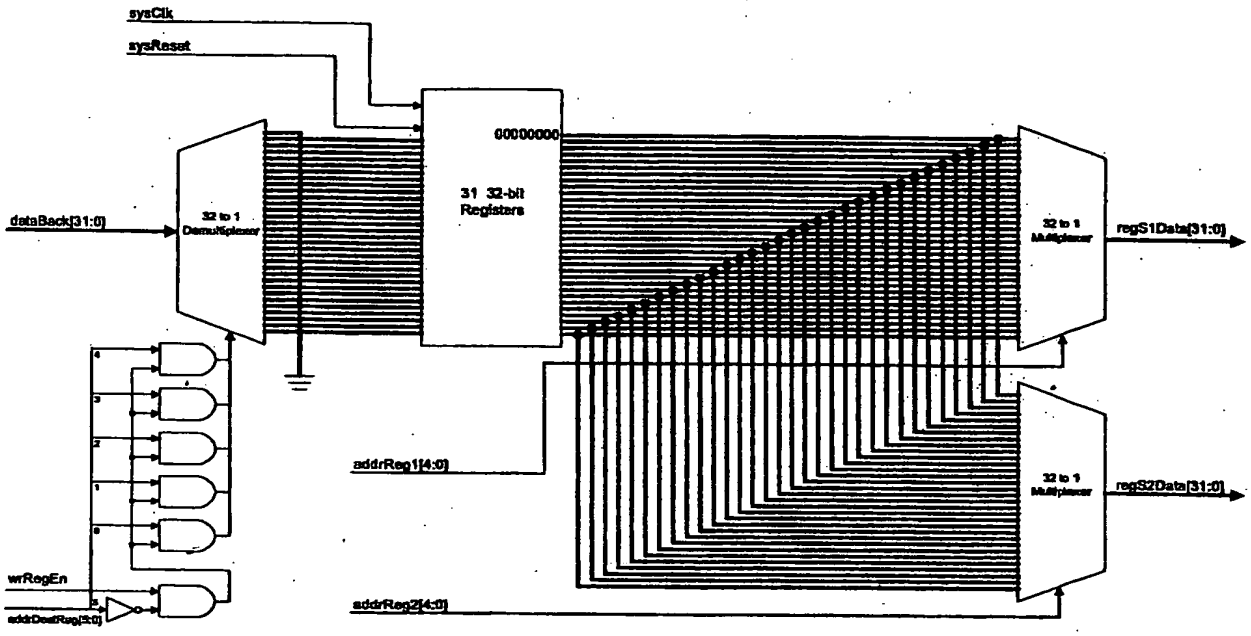

sysClk
sysReset
00000000
31 32-bit
Registers
dataBack[31:0]
32 to 1
Demultiplexer
32 to 1
Multiplexer
regS1Data[31:0]
addrReg1[4:0]
32 to 1
Multiplexer
regS2Data[31:0]
wrRegEn
addrReg2[4:0]

Fig. 10

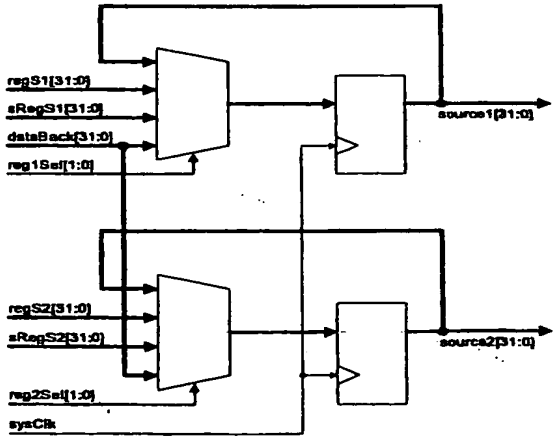

regS1[31:0]
sRegS1[31:0]
dataBack[31:0]
reg1Sel[1:0]
source1[31:0]
regS2[31:0]
sRegS2[31:0]
reg2Sel[1:0]
sysClk
source2[31:0]

Fig. 11